CAN

NI-CAN™ User Manual

May 2001 Edition
Part Number 370290B-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838,
China (ShenZhen) 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Malaysia 603 9596711,
Mexico 5 280 7625, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

Copyright © 1996, 2001 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The NI-CAN™ is warranted against defects in materials and workmanship for a period of 90 days from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

CVI™, LabVIEW™, National Instruments™, NI™, NI-CAN™, ni.com™, PXI™, and RTSI™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

The product described in this manual may be protected by one or more U.S. patents: U.S. Patent No. 5,938,754.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Compliance

FCC/Canada Radio Frequency Interference Compliance*

Determining FCC Class

The Federal Communications Commission (FCC) has rules to protect wireless communications from interference. The FCC places digital electronics into two classes. These classes are known as Class A (for use in industrial-commercial locations only) or Class B (for use in residential or commercial locations). Depending on where it is operated, this product could be subject to restrictions in the FCC rules. (In Canada, the Department of Communications (DOC), of Industry Canada, regulates wireless interference in much the same way.)

Digital electronics emit weak signals during normal operation that can affect radio, television, or other wireless products. By examining the product you purchased, you can determine the FCC Class and therefore which of the two FCC/DOC Warnings apply in the following sections. (Some products may not be labeled at all for FCC; if so, the reader should then assume these are Class A devices.)

FCC Class A products only display a simple warning statement of one paragraph in length regarding interference and undesired operation. Most of our products are FCC Class A. The FCC rules have restrictions regarding the locations where FCC Class A products can be operated.

FCC Class B products display either a FCC ID code, starting with the letters **EXN**, or the FCC Class B compliance mark that appears as shown here on the right.

Consult the FCC web site `http://www.fcc.gov` for more information.

FCC/DOC Warnings

This equipment generates and uses radio frequency energy and, if not installed and used in strict accordance with the instructions in this manual and the CE Mark Declaration of Conformity**, may cause interference to radio and television reception. Classification requirements are the same for the Federal Communications Commission (FCC) and the Canadian Department of Communications (DOC).

Changes or modifications not expressly approved by National Instruments could void the user's authority to operate the equipment under the FCC Rules.

Class A

Federal Communications Commission

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Canadian Department of Communications

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

Class B

Federal Communications Commission

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Canadian Department of Communications

This Class B digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe B respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

Compliance to EU Directives

Readers in the European Union (EU) must refer to the Manufacturer's Declaration of Conformity (DoC) for information** pertaining to the CE Mark compliance scheme. The Manufacturer includes a DoC for most every hardware product except for those bought for OEMs, if also available from an original manufacturer that also markets in the EU, or where compliance is not required as for electrically benign apparatus or cables.

To obtain the DoC for this product, click **Declaration of Conformity** at `ni.com/hardref.nsf/`. This website lists the DoCs by product family. Select the appropriate product family, followed by your product, and a link to the DoC appears in Adobe Acrobat format. Click the Acrobat icon to download or read the DoC.

* Certain exemptions may apply in the USA, see FCC Rules §15.103 **Exempted devices**, and §15.105(c). Also available in sections of CFR 47.

** The CE Mark Declaration of Conformity will contain important supplementary information and instructions for the user or installer.

Contents

About This Manual

How To Use the Manual Set xiii
Conventions xiii
Related Documentation xiv

Chapter 1
Introduction

CAN Overview 1-1
- History and Use of CAN 1-1
- CAN Identifiers and Message Priority 1-2
- CAN Frames 1-3
  - Start of Frame (SOF) 1-3
  - Arbitration ID 1-4
  - Remote Transmit Request (RTR) 1-4
  - Identifier Extension (IDE) 1-4
  - Data Length Code (DLC) 1-4
  - Data Bytes 1-4
  - Cyclic Redundancy Check (CRC) 1-4
  - Acknowledgment Bit (ACK) 1-5
  - End of Frame 1-5
- CAN Error Detection and Confinement 1-5
  - Error Detection 1-5
  - Error Confinement 1-6
- Low-Speed CAN 1-8

NI-CAN Hardware Overview 1-9
NI-CAN Software Overview 1-11
- Independent Design 1-11
- Object-Oriented Design 1-11
- NI-CAN Object Hierarchy 1-12
- NI-CAN Software Components 1-14
  - NI-CAN Driver and Utilities 1-14
  - Firmware Image Files 1-15
  - Language Interface Files 1-15
  - Application Examples 1-15
  - Interaction of Software Components with Your Application 1-16

RTSI Bus Overview 1-16
- The RTSI Solution 1-16
- Synchronizing with NI-DAQ 1-17

Chapter 2
Developing Your Application

Choosing Your Programming Method 2-1
Choosing a Method to Access the NI-CAN Software 2-1
(LabVIEW) Function Library 2-1
LabVIEW Real-Time (RT) 2-1
C/C++ Language Interfaces 2-2
Direct Entry Access 2-3
Choosing Which NI-CAN Objects to Use 2-4
Using CAN Network Interface Objects 2-4
Using CAN Objects 2-5
Programming Model for NI-CAN Applications 2-7
Step 1. Configure Objects 2-9
Step 2. Open Objects 2-9
Step 3. Start Communication 2-9
Step 4. Communicate Using Objects 2-9
Step 5. Close Objects 2-10
Checking Status of Function Calls 2-10
Checking Status in LabVIEW 2-11
Checking Status in C or C++ 2-12

Chapter 3
NI-CAN Programming Techniques

Using Queues 3-1
State Transitions 3-1
Empty Queues 3-2
Full Queues 3-2
Disabling Queues 3-2
Using the CAN Network Interface Object with CAN Objects 3-2
Detecting State Changes 3-4

Chapter 4
Application Examples

LabVIEW Examples 4-1
C/C++ Examples 4-1
Other Programming Languages 4-1

Chapter 5
NI-CAN Configuration and Diagnostic Utilities

Overview....5-1
Starting the NI-CAN Configuration Utility in Windows Me/98/95....5-1
Starting the NI-CAN Configuration Utility in Windows 2000/NT....5-2
Starting the NI-CAN Remote Configuration Utility for LabVIEW RT....5-3
Starting the NI-CAN Diagnostic Utility....5-3

Appendix A
Windows Me/98/95: Troubleshooting and Common Questions

Appendix B
Windows NT: Troubleshooting and Common Questions

Appendix C
Windows 2000: Troubleshooting and Common Questions

Appendix D
Cabling Requirements for High-Speed CAN

Appendix E
Cabling Requirements for Low-Speed CAN

Appendix F
Cabling Requirements for Dual-Speed CAN

Appendix G
RTSI Bus

Appendix H
Specifications

Appendix I
Technical Support Resources

Glossary

Index

Figures

Figure 1-1. Example of CAN Arbitration 1-3
Figure 1-2. Standard and Extended Frame Formats 1-3
Figure 1-3. Simple CAN Device Network Application 1-12
Figure 1-4. Applying NI-CAN Objects to the Example in Figure 1-3 1-13
Figure 1-5. Interaction of NI-CAN Software Components 1-16

Figure 2-1. General Program Steps Using NI-CAN Functions 2-8

Figure 3-1. Flowchart for CAN Frame Reception 3-3

Figure 5-1. NI-CAN Diagnostic Utility after Testing 5-4

Figure A-1. CAN Interface That Is Not Working Properly A-2

Figure C-1. CAN Interface That Is Not Working Properly C-2
Figure C-2. CAN Interface That has Not Been Recognized Properly C-3

Figure D-1. Pinout for 9-Pin D-Sub Connector D-1
Figure D-2. Pinout for 5-Pin Combicon-Style Pluggable Screw Terminal D-2
Figure D-3. PCMCIA-CAN Cable D-2
Figure D-4. AT-CAN/2 Parts Locator Diagram D-3
Figure D-5. PCI-CAN/2 Parts Locator Diagram D-4
Figure D-6. PXI-8461 Parts Locator Diagram D-5
Figure D-7. Power Source Jumpers D-6
Figure D-8. Termination Resistor Placement D-8
Figure D-9. Cabling Example D-9

Figure E-1. Pinout for 9-Pin D-Sub Connector E-1
Figure E-2. PCMCIA-CAN/LS Cable E-2
Figure E-3. PCI-CAN/LS2 Parts Locator Diagram E-3
Figure E-4. PXI-8460 Parts Locator Diagram E-4
Figure E-5. Power Source Jumpers E-5
Figure E-6. Termination Resistor Placement for Low-Speed CAN E-6
Figure E-7. Location of Termination Resistors on PCI-CAN/LS2 Board E-9
Figure E-8. Preparing Lead Wires of Replacement Resistors E-10
Figure E-9. Location of Termination Resistors on a PXI-8460 ..... E-11
Figure E-10. Preparing Lead Wires of Replacement Resistors E-11

Figure E-11. Preparing Lead Wires of PCMCIA-CAN/LS Cable Replacement Resistors E-12
Figure E-12. Cabling Example E-13

Figure G-1. AT-CAN Series RTSI Connector Pinout G-1
Figure G-2. PCI-CAN Series RTSI Connector Pinout G-2

Tables

Table 2-1. NI-CAN Error Cluster 2-11
Table 2-2. NI-CAN Status Code 2-12

Table D-1. Power Requirements for the CAN Physical Layer for Bus-Powered Versions D-6
Table D-2. ISO 11898 Specifications for Characteristics of a CAN_H and CAN_L Pair of Wires D-7
Table D-3. DeviceNet Cable Length Specifications D-7

Table E-1. Power Requirements for the Low-Speed CAN Physical Layer for Bus-Powered Versions E-5
Table E-2. ISO 11519-2 Specifications for Characteristics of a CAN_H and CAN_L Pair of Wires E-6

Table G-1. Pins Used By the PXI-846*x* Series Boards G-3

About This Manual

This manual describes the features of the CAN Hardware and NI-CAN software. It assumes that you are already familiar with the operating system you are using.

How To Use the Manual Set

Use the *Installation Guide, CAN Hardware and the NI-CAN Software for Windows 2000/NT/Me/9*x in the jewel case of your program CD to install and configure your CAN hardware and the NI-CAN software.

Use this *NI-CAN User Manual* to learn the basics of CAN and how to develop an application program.

Use the *NI-CAN Programmer Reference Manual* for specific information about each NI-CAN function and object, such as format, description and and parameters.

Conventions

The following conventions appear in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories,

programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

Platform Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- ANSI/ISO Standard 11898-1993, *Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*
- ANSI/ISO Standard 11519-1, 2 *Road Vehicles—Low Speed Serial Data Communications*, Part 1 and 2
- *CAN Specification Version 2.0*, 1991, Robert Bosch Gmbh., Postfach 500, D-7000 Stuttgart 1
- CiA Draft Standard 102, Version 2.0, CAN Physical Layer for Industrial Applications
- CompactPCI Specification, Revision 2.0, PCI Industrial Computers Manufacturers Group
- DeviceNet Specification, Version 2.0, Open DeviceNet Vendor Association
- *PXI Specificatio*n, Revision 1.0, National Instruments Corporation
- LabVIEW Online Reference
- Measurement and Automation Explorer (MAX) Online Reference
- Microsoft Win32 Software Development Kit (SDK) online help

1

Introduction

This chapter gives an overview of CAN and the NI-CAN hardware and NI-CAN software.

CAN Overview

History and Use of CAN

In the past few decades, the need for improvements in automotive technology has led to increased use of electronic control systems for functions such as engine timing, anti-lock brake systems, and distributorless ignition. With conventional wiring, data is exchanged in these systems using dedicated signal lines. As the complexity and number of devices has increased, using dedicated signal lines has become increasingly difficult and expensive.

To overcome the limitations of conventional automotive wiring, Bosch developed the Controller Area Network (CAN) in the mid-1980s. Using CAN, devices (controllers, sensors, and actuators) are connected on a common serial bus. This network of devices can be thought of as a scaled down, real-time, low cost version of networks used to connect personal computers. Any device on a CAN network can communicate with any other device using a common pair of wires.

As CAN implementations increased in the automotive industry, CAN was standardized internationally as ISO 11898, and CAN chips were created by major semiconductor manufacturers such as Intel, Motorola, and Phillips. With these developments, many manufacturers of industrial automation equipment began to consider CAN for use in industrial applications. Comparison of the requirements for automotive and industrial device networks showed many similarities, including the transition away from dedicated signal lines, low cost, resistance to harsh environments, and high real-time capabilities.

Because of these similarities, CAN became widely used in industrial applications such as textile machinery, packaging machines, and production line equipment such as photoelectric sensors and motion

controllers. By the mid-1990s, CAN was specified as the basis of many industrial device networking protocols, including DeviceNet, CANopen, and Smart Distributed System (SDS).

With its growing popularity in automotive and industrial applications, CAN has been increasingly used in a wide variety of diverse applications. Use in systems such as agricultural equipment, nautical machinery, medical apparatus, semiconductor manufacturing equipment, and machine tools testify to the incredible versatility of CAN.

CAN Identifiers and Message Priority

When a CAN device transmits data onto the network, an identifier that is unique throughout the network precedes the data. The identifier defines not only the content of the data, but also the priority. A CAN identifier, along with its associated data, is often referred to as a *CAN Object*.

When a device transmits a message onto the CAN network, all other devices on the network receive that message. Each receiving device performs an acceptance test on the identifier to determine if the message is relevant to it. If the received identifier is not relevant to the device (such as RPM received by an air conditioning controller), the device ignores the message.

When more than one CAN device transmits a message simultaneously, the identifier is used as a priority to determine which device gains access to the network. The lower the numerical value of the identifier, the higher its priority.

Figure 1-1 shows two CAN devices attempting to transmit messages, one using identifier 647 hex, and the other using identifier 6FF hex. As each device transmits the 11 bits of its identifier, it examines the network to determine if a higher-priority identifier is being transmitted simultaneously. If an identifier collision is detected, the losing device(s) immediately cease transmission, and wait for the higher-priority message to complete before automatically retrying. Because the highest priority identifier continues its transmission without interruption, this scheme is referred to as *nondestructive bitwise arbitration*, and CAN's identifier is often referred to as an *arbitration ID*. This ability to resolve collisions and continue with high-priority transmissions is one feature that makes CAN ideal for real-time applications.

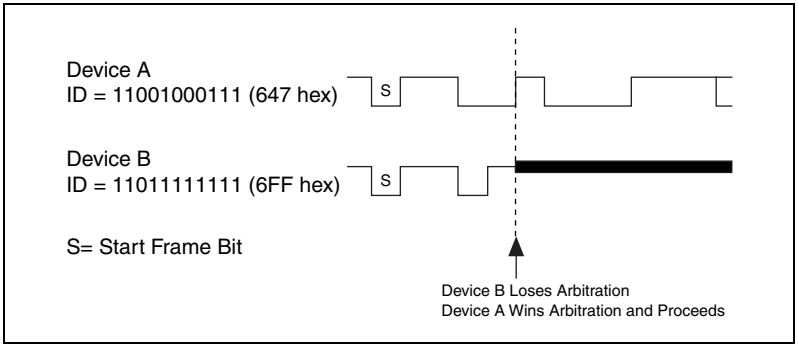


Figure 1-1. Example of CAN Arbitration

CAN Frames

In a CAN network, the messages transferred across the network are called frames. The CAN protocol supports two frame formats as defined in the Bosch version 2.0 specifications, the essential difference being in the length of the arbitration ID. In the standard frame format (also known as 2.0A), the length of the ID is 11 bits. In the extended frame format (also known as 2.0B), the length of the ID is 29 bits. The ISO 11898 specification supports only the standard frame format. Figure 1-2 shows the essential fields of the standard and extended frame formats, and the following sections describe each field.

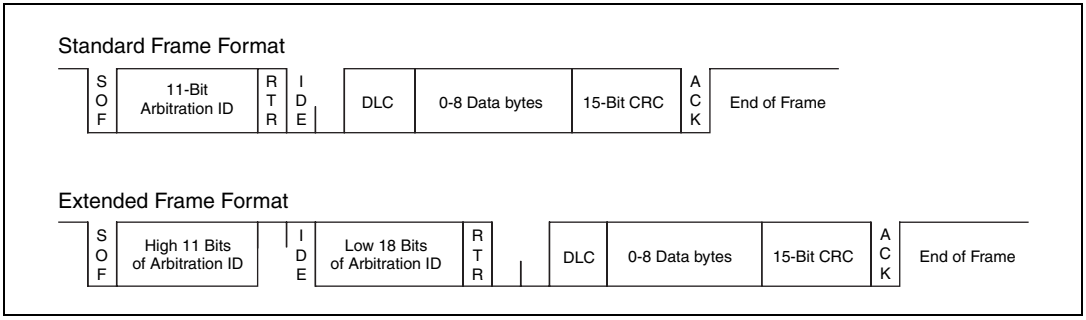


Figure 1-2. Standard and Extended Frame Formats

Start of Frame (SOF)

Start of Frame is a single bit (0) that marks the beginning of a CAN frame.

Arbitration ID

The arbitration ID fields contain the identifier for a CAN frame. The standard format has one 11-bit field, and the extended format has two fields, which are 11 and 18 bits in length. In both formats, bits of the arbitration ID are transmitted from high to low order.

Remote Transmit Request (RTR)

The Remote Transmit Request bit is dominant (0) for data frames, and recessive (1) for remote frames. Data frames are the fundamental means of data transfer on a CAN network, and are used to transmit data from one device to one or more receivers. A device transmits a remote frame to request transmission of a data frame for the given arbitration ID. The remote frame is used to request data from its source device, rather than waiting for the data source to transmit the data on its own.

Identifier Extension (IDE)

The Identifier Extension bit differentiates standard frames from extended frames. Because the IDE bit is dominant (0) for standard frames and recessive (1) for extended frames, standard frames are always higher priority than extended frames.

Data Length Code (DLC)

The Data Length Code is a 4-bit field that indicates the number of data bytes in a data frame. In a remote frame, the Data Length Code indicates the number of data bytes in the requested data frame. Valid Data Length Codes range from zero to eight.

Data Bytes

For data frames, this field contains from 0 to 8 data bytes. Remote CAN frames always contain zero data bytes.

Cyclic Redundancy Check (CRC)

The 15-bit Cyclic Redundancy Check detects bit errors in frames. The transmitter calculates the CRC based on the preceding bits of the frame, and all receivers recalculate it for comparison. If the CRC calculated by a receiver differs from the CRC in the frame, the receiver detects an error.

Acknowledgment Bit (ACK)

All receivers use the Acknowledgment Bit to acknowledge successful reception of the frame. The ACK bit is transmitted recessive (1), and is overwritten as dominant (0) by all devices that receive the frame successfully. The receivers acknowledge correct frames regardless of the acceptance test performed on the arbitration ID. If the transmitter of the frame detects no acknowledgment, it could mean that the receivers detected an error (such as a CRC error), the ACK bit was corrupted, or there are no receivers (for example, only one device on the network). In such cases, the transmitter automatically retransmits the frame.

End of Frame

Each frame ends with a sequence of recessive bits. After the required number of recessive bits, the CAN bus is idle, and the next frame transmission can begin.

CAN Error Detection and Confinement

One of the most important and useful features of CAN is its high reliability, even in extremely noisy environments. CAN provides a variety of mechanisms to detect errors in frames. This error detection is used to retransmit the frame until it is received successfully. CAN also provides an error confinement mechanism used to remove a malfunctioning device from the CAN network when a high percentage of its frames result in errors. This error confinement prevents malfunctioning devices from disturbing the overall network traffic.

Error Detection

Whenever any CAN device detects an error in a frame, that device transmits a special sequence of bits called an error flag. This error flag is normally detected by the device transmitting the invalid frame, which then retransmits to correct the error. The retransmission starts over from the start of frame, and thus arbitration with other devices is again possible.

CAN devices detect the following errors, which are described in the following sections:

- Bit error
- Stuff error
- CRC error

- Form error
- Acknowledgment error

Bit Error

During frame transmissions, a CAN device monitors the bus on a bit-by-bit basis. If the bit level monitored is different from the transmitted bit, a bit error is detected. This bit error check applies only to the Data Length Code, Data Bytes, and Cyclic Redundancy Check fields of the transmitted frame.

Stuff Error

Whenever a transmitting device detects five consecutive bits of equal value, it automatically inserts a complemented bit into the transmitted bit stream. This stuff bit is automatically removed by all receiving devices. The bit stuffing scheme is used to guarantee enough edges in the bit stream to maintain synchronization within a frame.

A stuff error occurs whenever six consecutive bits of equal value are detected on the bus.

CRC Error

A CRC error is detected by a receiving device whenever the calculated CRC differs from the actual CRC in the frame.

Form Error

A form error occurs when a violation of the fundamental CAN frame encoding is detected. For example, if a CAN device begins transmitting the Start Of Frame bit for a new frame before the End Of Frame sequence completes for a previous frame (does not wait for bus idle), a form error is detected.

Acknowledgment Error

An acknowledgment error is detected by a transmitting device whenever it does not detect a dominant Acknowledgment Bit (ACK).

Error Confinement

To provide for error confinement, each CAN device must implement a transmit error counter and a receive error counter. The transmit error counter is incremented when errors are detected for transmitted frames, and decremented when a frame is transmitted successfully. The receive error counter is used for received frames in much the same way. The error

counters are increased more for errors than they are decreased for successful reception/transmission. This ensures that the error counters will generally increase when a certain ratio of frames (roughly 1/8) encounter errors. By maintaining the error counters in this manner, the CAN protocol can generally distinguish temporary errors (such as those caused by external noise) from permanent failures (such as a broken cable). For complete information on the rules used to increment/decrement the error counters, refer to the CAN specification (ISO 11898).

With regard to error confinement, each CAN device may be in one of three states: error active, error passive, and bus off.

Error Active State

When a CAN device is powered on, it begins in the error active state. A device in error active state can normally take part in communication, and transmits an active error flag when an error is detected. This active error flag (sequence of dominant 0 bits) causes the current frame transmission to abort, resulting in a subsequent retransmission. A CAN device remains in the error active state as long as the transmit and receive error counters are both below 128. In a normally functioning network of CAN devices, all devices are in the error active state.

Error Passive State

If either the transmit error counter or the receive error counter increments above 127, the CAN device transitions into the error passive state. A device in error passive state can still take part in communication, but transmits a passive error flag when an error is detected. This passive error flag (sequence of recessive 1 bits) generally does not abort frames transmitted by other devices. Since passive error flags cannot prevail over any activity on the bus line, they are noticed only when the error passive device is transmitting a frame. Thus, if an error passive device detects a receive error on a frame which is received successfully by other devices, the frame is not retransmitted.

One special rule to keep in mind is that when an error passive device detects an acknowledgment error, it does not increment its transmit error counter. Thus, if a CAN network consists of only one device (for example, if you do not connect a cable to your National Instruments CAN interface), and that device attempts to transmit a frame, it retransmits continuously but never goes into bus off state (although it eventually reaches error passive state).

Bus Off State

If the transmit error counter increments above 255, the CAN device transitions into the bus off state. A device in the bus off state does not transmit or receive any frames, and thus cannot have any influence on the bus. The bus off state is used to disable a malfunctioning CAN device which frequently transmits invalid frames, so that the device does not adversely impact other devices on the network. When a CAN device has transitioned to bus off, it can be placed back into error active state (with both counters reset to zero) only by manual intervention. For sensor/actuator types of devices, this often involves powering the device off then on. For NI-CAN network interfaces, communication can be started again using a function such as `ncAction`.

Low-Speed CAN

Low-speed CAN is commonly used to control “comfort” devices in an automobile, such as seat adjustment, mirror adjustment, and door locking. It differs from “high-speed” CAN in that the maximum baud rate is 125K and it utilizes CAN transceivers that offer fault-tolerant capability. This enables the CAN bus to keep operating even if one of the wires is cut or short-circuited because it operates on relative changes in voltage, and thus provides a much higher level of safety. The fault tolerance feature means that communications capability is maintained even if any of the ISO 11519 wiring failures occur. The transceiver solves many common and frequent wiring problems such as poor connectors, and also overcomes short circuits of either transmission wire to ground or battery voltage, or the other transmission wire. The transceiver resolves the fault situation without involvement of external hardware or software. On the detection of a fault, the transceiver switches to a one wire transmission mode and automatically switches back to differential mode if the fault is removed.

Special resistors are added to the circuitry for the proper operation of the fault-tolerant transceiver. The values of the resistors depend on the number of nodes and the resistance values per node. For guidelines on selecting the resistor, refer to, Appendix E, *Cabling Requirements for Low-Speed CAN*, in this manual.

Because the low-speed transceiver switches to a fault tolerant mode on fault detection and continues to maintain communications, NI-CAN provides a special attribute, `NC_ATTR_LOG_COMM_ERRS`, which when set to `NC_TRUE` enables the reporting of such warnings in the Read queue of the Network interface rather than in the Status returned from a function call. The default value of this attribute is `NC_FALSE`, which enables the reporting of low-speed transceiver warnings in the Status returned from a function call.

Refer to the CAN network interface object attributes section in the *NI-CAN Programmer Reference Manual* for details on how to use this attribute.

NI-CAN Hardware Overview

The National Instruments CAN hardware covered in this manual includes the AT-CAN and AT-CAN/2 (**Windows Me/9*x* only**), PCI-CAN, PCI-CAN/2, PCI-CAN/LS (low-speed CAN), PCI-CAN/LS2, PCI-CAN/DS (dual-speed CAN), PCMCIA-CAN, PCMCIA-CAN/2, PXI-8460 (low-speed: one or two port), PXI-8461 (high-speed: one or two port) and PXI-8462 (dual-speed: port one high-speed, port two low-speed).

The AT-CAN series boards are fully software configurable and compliant with the Plug and Play ISA standard. With an AT-CAN or AT-CAN/2 board, you can make your PC AT-compatible computer communicate with and control CAN devices.

The PCI-CAN, PCI-CAN/LS and PCI-CAN/DS series boards are completely software configurable and compliant with the PCI Local Bus Specification. With a PCI-CAN, PCI-CAN/LS or PCI-CAN/DS series board, you can make your PC-compatible computer with PCI Local Bus slots communicate with and control CAN devices.

The PCMCIA-CAN series cards are Type II PC Cards that are completely software configurable and compliant with the PCMCIA standards for 16-bit PC Cards. With a PCMCIA-CAN series card, you can make your PC-compatible notebook with PCMCIA sockets communicate with and control CAN devices.

The PXI-8460, PXI-8461, and PXI-8462 are software configurable and compliant with the *PXI Specification* and *CompactPCI Specification*. With a PXI-846*x* board, you can make your PXI or CompactPCI chassis communicate with and control CAN devices.

The CAN hardware supports a wide variety of transfer rates up to 1 Mb/s. CAN interfacing is accomplished using the Intel 82527 CAN controller chip. The high-speed CAN physical layer fully conforms to the ISO 11898 physical layer specification for CAN and is optically isolated to 500 V. The low-speed CAN physical layer conforms to the ISO 11519-2 physical layer specification for CAN and is also optically isolated to 500 V.

AT-CAN, PCI-CAN, and PXI-8461 series boards are available with two physical connector types: DB-9 D-Sub and Combicon-style pluggable

screw terminals. Low-speed PCI-CAN/LS, PCI-CAN/DS, PXI-8460, and PXI-8462 boards are available with DB-9 D-Sub connectors. PCMCIA-CAN, PCMCIA-CAN/LS and PCMCIA-CAN/DS cables include both a DB-9 D-Sub and a pluggable screw terminal.

The CAN physical layer on AT-CAN, PCI-CAN and PXI-846*x* series cards can be powered either internally (from the card) or externally (from the bus cable power). The power source for the CAN physical layer for each port is configured with a jumper.

There are four types of cables available for the PCMCIA-CAN cards:

1. PCMCIA-CAN bus powered transceiver cables. The CAN physical layer is powered externally (from the bus cable power).
2. PCMCIA-CAN internally powered transceiver cables. The CAN physical layer is powered internally (from the card).
3. PCMCIA-CAN/LS cables. The low-speed CAN physical layer and the V-BAT pin of the low-speed transceiver are powered internally. This cable also requires that only the V-, CAN_L and CAN_H be connected to the bus.
4. PCMCIA-CAN/DS cables. The high-speed port (port 1) physical layer is powered internally. The low-speed port (port 2) physical layer is identical to the PCMCIA-CAN/LS cable.

The PXI-846*x*, PCI-CAN and AT-CAN series boards use the Real-Time System Integration (RTSI) bus to solve the problem of synchronizing several functions across multiple boards to a common trigger or timing event. For PCI-CAN and AT-CAN, the RTSI bus consists of the National Instruments RTSI bus interface and ribbon cable to route timing and trigger signals between the CAN hardware and National Instruments DAQ, IMAQ, or additional CAN hardware. For the PXI-846*x*, the RTSI bus is implemented by using the National Instruments PXI trigger bus to route timing and trigger signals between the CAN hardware and National Instruments DAQ, IMAQ, or additional CAN hardware. Although the PXI-846*x* series board with RTSI bus is available in a PXI chassis, there important issues to consider when using it in a compactPCI chassis. Refer to Appendix G, *RTSI Bus*, in this manual for detailed information about the RTSI interface. Also see the *RTSI Bus Overview* and *The RTSI Solution* sections later in this chapter.

All of the CAN hardware uses the Intel 386EX embedded processor to implement time-critical features provided by the NI-CAN software. The CAN hardware communicates with the NI-CAN driver through on-board shared memory and an interrupt.

NI-CAN Software Overview

Independent Design

The NI-CAN Application Programming Interface (API), like most National Instruments APIs, is largely independent of operating system and programming language. You can use NI-CAN in a wide variety of programming environments, including LabVIEW and C programming environments such as LabWindows/CVI. Applications written for NI-CAN are also portable across different operating systems, such as Windows 2000/NT and Windows Me/98/95.

In addition to being independent of operating system and programming language, NI-CAN is designed to be largely independent of a specific device network protocol. Device network independence means that where possible, terminology specific to CAN alone is avoided so that the API can be expanded later to support higher level protocols based on CAN. Examples of such protocols include DeviceNet, Smart Distributed System (SDS), and CANopen. Device network independence largely applies to terminology such as function names, and in no way limits access to the CAN network. For example, the function provided to read data from a CAN frame is called `ncRead`, as opposed to a name specific to CAN, such as `ncReadCanFrame`.

Object-Oriented Design

NI-CAN often uses object-oriented terminology and concepts. Object-oriented terminology provides an excellent model for describing device networks in terms that are easy to understand.

In object-oriented terminology, the term *class* describes a classification of an object, and the term *instance* refers to a specific instance of that object. The term *object* is generally used as a synonym for instance. For example, NI-CAN defines a class called the CAN Network Interface Object, which encapsulates any network interface port on a National Instruments CAN hardware product. Specific instances of the CAN Network Interface Object are referenced with names like `CAN0` and `CAN1`. Each instance of a particular class has *attributes* that define its externally visible qualities, as well as *methods* that are used to perform actions. For example, each instance of the CAN Network Interface Object has an attribute for the baud rate (bits per second) used for communication, as well as a method used to transmit CAN frames onto the network.

For more information on object-oriented and CAN terminology, refer to the *Glossary*.

NI-CAN Object Hierarchy

The basic model of the NI-CAN software architecture is a hierarchical collection of objects (instances), each of which has attributes and methods. The hierarchy shows relationships between various objects. In general, a given object in the hierarchy has an "is used to access" relationship with all objects above it in the hierarchy.

As an example, consider a CAN device network in which the network interface of a host computer is physically connected to two devices, a pushbutton and an LED, as shown in Figure 1-3.

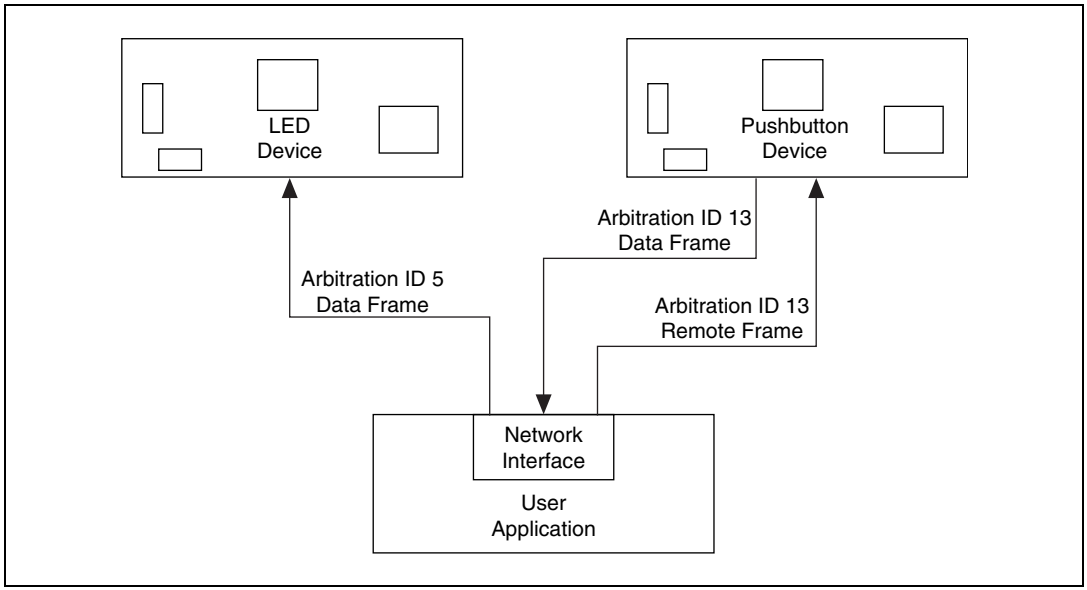


Figure 1-3. Simple CAN Device Network Application

The pushbutton device transmits the state of the button in a CAN data frame with standard arbitration ID 13. The frame data consists of a single byte—zero if the button is off, one if the button is on. For an NI-CAN application to obtain the current state of the pushbutton, it transmits a CAN remote frame with standard arbitration ID 13. The pushbutton device responds to this remote transmission request by transmitting the button state in its CAN data frame.

The LED device expects to obtain the state of the LED from a CAN data frame with standard arbitration ID 5. It expects the frame data to consist of a single byte—zero to turn the light off, one to turn the light on.

Figure 1-4 shows how NI-CAN objects encapsulate access to this CAN device network. The ovals in Figure 1-4 indicate NI-CAN objects, and the dotted lines indicate what each object encapsulates.

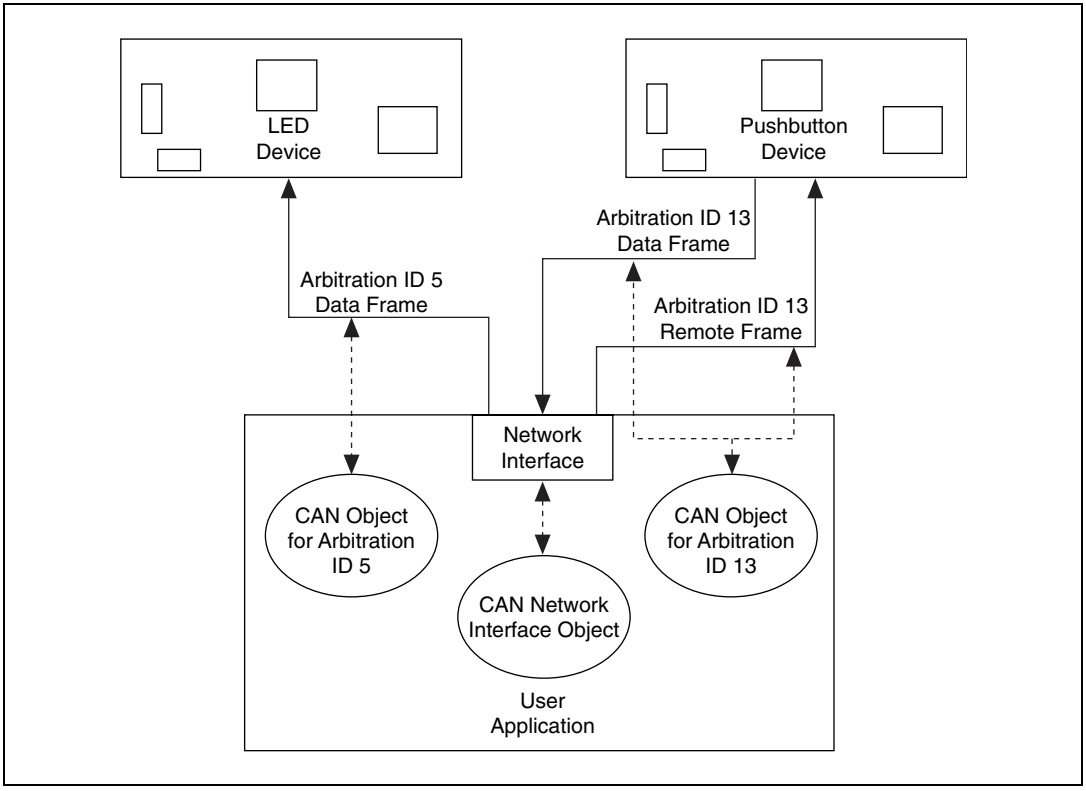


Figure 1-4. Applying NI-CAN Objects to the Example in Figure 1-3

The CAN Network Interface Object encapsulates the entire CAN network interface. Its attributes configure settings that apply to the network interface as a whole. For example, the CAN Network Interface Object contains an attribute you can use to set the baud rate that the network interface hardware uses for communication. You can also use the CAN Network Interface Object to communicate on the CAN device network. For example, you can use the write function to transmit a CAN remote frame to the pushbutton device, then use the read function to retrieve the resulting CAN data frame. Because the CAN Network Interface Object provides direct access to the

network interface, the write and read functions require all information about the CAN frame to be specified, including arbitration ID, whether the frame is a CAN data frame or a CAN remote frame, the number of data bytes, and the frame data (assuming a CAN data frame).

The CAN Object encapsulates a specific arbitration ID, along with its data. In addition to providing the ability to transmit and receive frames for a specific arbitration ID, CAN Objects also provide various forms of background access. For example, you can configure a CAN Object for arbitration ID 13 (the pushbutton) to automatically transmit a CAN remote frame every 500 ms, and to store the data of the resulting CAN data frame for later retrieval. After the CAN Object is configured in this manner, you can use the read function to obtain a single data byte that holds the most recent state of the pushbutton.

NI-CAN Software Components

The following section highlights important components of the NI-CAN software, and describes the function of each component.

NI-CAN Driver and Utilities

- A documentation file, `readme.txt`, contains important information about the NI-CAN software and a description of any new features. Before you use the software, read this file for the most recent information.
- A 32-bit, multitasking aware device driver is used to interface with National Instruments CAN hardware. Under Windows Me/98/95, this is a dynamically loadable, Plug and Play aware virtual device driver (VxD). Under Windows 2000/NT, this is a native Windows 2000/NT kernel driver.
- A Win32 dynamic link library, `nican.dll`, acts as the interface between all CAN applications and the NI-CAN device driver.
- The NI-CAN Configuration utility is used to modify the configuration of the NI-CAN software. Under Windows Me/98/95, this utility is integrated into the Windows Device Manager. Under Windows 2000/NT, this utility is a control panel application.
- The NI-CAN Diagnostic utility is used to verify that the CAN hardware and software have been installed properly.
- The NI-CAN Remote Configuration utility is used to configure and test NI-CAN software on a remote LabVIEW Real-Time system. This utility communicates with a LabVIEW Real-Time (RT) System (PXI chassis) using TCP/IP (Ethernet). The utility is used to configure

the NI-CAN software and verify that the CAN hardware and software have been installed properly. This utility is used only for LabVIEW RT systems. To configure and diagnose NI-CAN installations on Windows, use the NI-CAN Configuration utility and NI-CAN Diagnostic utility.

Firmware Image Files

All National Instruments CAN hardware products contain an on-board microprocessor. This microprocessor is used so that all time-critical aspects of the NI-CAN software can be executed separately from your Windows application. The firmware image which runs on the on-board microprocessor, `nican.nfw`, is loaded and executed automatically when your NI-CAN application starts.

Language Interface Files

- A documentation file, `readme.txt`, contains information about the NI-CAN language interface files.
- A 32-bit C language include file, `nican.h`, contains NI-CAN function prototypes, host data types, and various predefined constants.
- A 32-bit C language interface file, `nicanmsc.lib`, is used by Microsoft C/C++ applications to access the NI-CAN DLL.
- A 32-bit C language interface file, `nicanbor.lib`, is used by Borland C/C++ (5.0 or greater) applications to access the NI-CAN DLL.
- A 32-bit C language interface file, `nican.lib`, is used by LabWindows/CVI applications to access the NI-CAN DLL. This file is installed in your LabWindows/CVI environment.
- NI-CAN function panels for LabWindows/CVI (`nican.fp`) enable you to develop a CAN application with LabWindows/CVI. These function panels are installed in your LabWindows/CVI environment.
- A 32-bit LabVIEW function library, `nican.llb`, is used by LabVIEW applications to access the NI-CAN DLL. This library and the associated palettes are installed in your LabVIEW environment.

Application Examples

The NI-CAN software includes several sample applications. For a description of the sample application files, refer to Chapter 4, *Application Examples*.

Interaction of Software Components with Your Application

Figure 1-5 shows the interaction between your application and the NI-CAN software components.

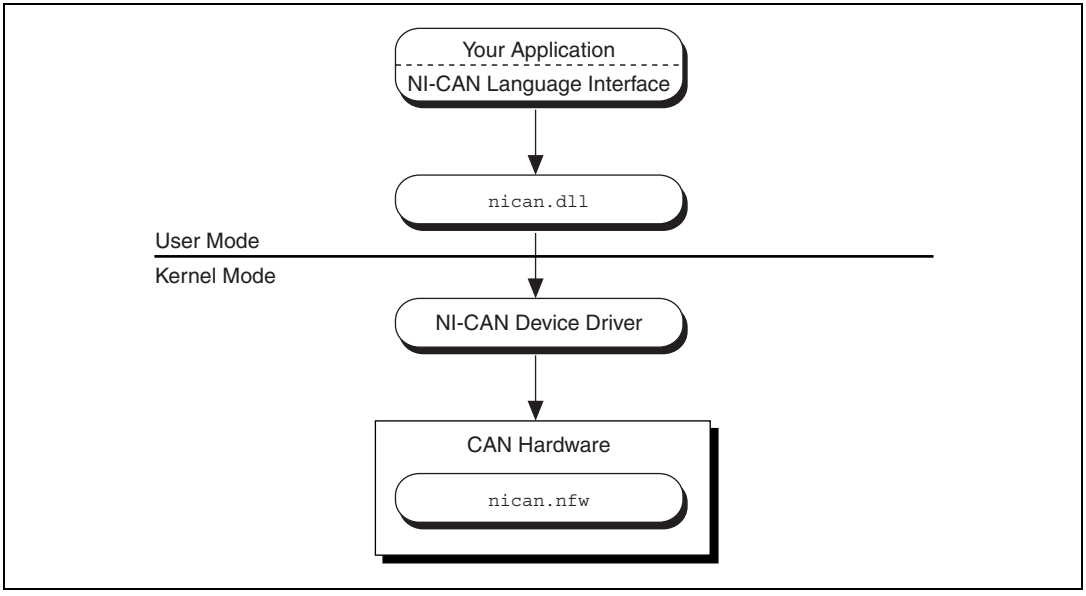


Figure 1-5. Interaction of NI-CAN Software Components

RTSI Bus Overview

RTSI is an acronym for Real-Time System Integration. It is the National Instruments timing bus that connects CAN and DAQ boards directly. This is done via connectors on top of the PCI-CAN and AT-CAN series boards, and the PXI trigger bus on the PXI-846*x* series boards, for precise synchronization of functions.

The RTSI Solution

A common problem with interface boards is that you cannot easily synchronize several functions across multiple boards to a common trigger or timing event. CAN boards use the RTSI bus to solve this problem.

For PCI-CAN and AT-CAN series boards, the RTSI bus consists of connecting the National Instruments RTSI bus interface with RTSI ribbon cable to route timing and trigger signals between the CAN board and other

National Instruments RTSI-equipped hardware. Refer to *RTSI Bus*, Appendix G in this manual, for detailed information about the PCI-CAN and AT-CAN series RTSI interfaces.

For the PXI-846*x* series CAN boards, the RTSI bus consists of using the National Instruments PXI trigger bus to route timing and trigger signals between the PXI-846*x* series board and other National Instruments RTSI-equipped PXI boards. Regarding the RTSI interface on your PXI-846*x* series board, there are important issues to consider when using it in a CompactPCI chassis. Refer to *RTSI Bus*, Appendix G in this manual, for detailed information about the PXI-846*x* series RTSI interface.

The RTSI bus allows you to synchronize a CAN board with multiple National Instruments DAQ, IMAQ, or additional CAN boards in your computer. The RTSI bus can also synchronize CAN bus events between multiple CAN boards. The trigger lines on the RTSI bus provide a flexible interconnection scheme between CAN boards as well as between National Instruments DAQ, IMAQ, and CAN boards.

Synchronizing with NI-DAQ

Recent advances in test applications, in the automotive industry for example, have demanded tighter integration of DAQ and CAN measurements. The physical quantity being measured by DAQ and CAN devices must be done close together (synchronized in time) to correlate the data. This synchronization can be done in software, but the latency of the operating system introduces delays that may not be acceptable for some test applications. National Instruments CAN DAQ boards are equipped with an RTSI bus that allows routing of timing/trigger signals between them. In a system coupled using the RTSI bus, a CAN or DAQ board can be the Master driving the timing/trigger signals, while other DAQ or CAN boards are Slaves to this timing signal and base their actions on this signal. Because the RTSI timing signals are handled in hardware, the host software running on the PC does not interact in the acquisition or transmission (other than retrieving the data when it has being acquired or writing new data).

Before using RTSI between CAN and DAQ, you must decide which board to use as the Master. NI-CAN software allows easy configuration of the Network Interface or CAN Objects as a Master or Slave. Note that both configurations can exist on the same board, but not on the same object.

The software attribute that configures the object as a Master or Slave is the RTSI Mode. The following RTSI Modes are available in NI-CAN:

- Disable RTSI

- On RTSI Input—Transmit CAN Frame
- On RTSI Input—Timestamp RTSI event
- RTSI Output on Receiving CAN Frame
- RTSI Output on Transmitting CAN Frame
- RTSI Output on `ncAction` call

The second and third modes listed above configure the object (Network Interface or CAN Object) as a Slave that takes an action on the RTSI signal. The last three modes configure the object (Network Interface or CAN Object) as a Master.

For more information on these modes, refer to the *NI-CAN Programmer Reference Manual*. For information on how to access examples in both LabVIEW and C/C++ that use RTSI, refer to Chapter 4, *Application Examples*.

2

Developing Your Application

This chapter explains how to develop an application program using the NI-CAN functions.

Choosing Your Programming Method

Choosing a Method to Access the NI-CAN Software

Applications can access the NI-CAN dynamic link library (`nican.dll`) either by using an NI-CAN language interface or by direct entry access.

(LabVIEW) Function Library

For applications written in LabVIEW (5.1 or later), the NI-CAN function library for LabVIEW (`nican.llb`) provides a LabVIEW function to access each NI-CAN function easily.

For a description of each NI-CAN function for LabVIEW, refer to the *NI-CAN Programmer Reference Manual*.

LabVIEW Real-Time (RT)

NI-CAN applications developed with LabVIEW RT (6.0.3 or later) provide greater determinism than a Windows environment can guarantee. LabVIEW RT combines the user-friendliness of LabVIEW with the power of real-time systems, so you can use graphical programming to create deterministic applications. Using a host PC or PXI system running a Windows operating system, you can create LabVIEW RT Virtual Instruments (VIs) by using the same NI-CAN function library (`nican.llb`) you use to create NI-CAN VIs in LabVIEW for Windows. The host PC is used to download applications to an RT Series device, such as a PXI controller running LabVIEW RT. VIs downloaded to the RT device run in real time. Refer to the *LabVIEW Real-Time Help*, available by selecting **Help»LabVIEW Real-Time Help** from the LabVIEW RT development environment for more information.

C/C++ Language Interfaces

You can use an NI-CAN C language interface if your application is written in Microsoft Visual C/C++ (5.0 or later), Borland C/C++ (5.0 or later), or LabWindows/CVI (4.0 or later) with Microsoft C. For other programming languages or development environments, you must access `nican.dll` directly.

To use a C/C++ language interface, include the `nican.h` header file in your code, then link the appropriate NI-CAN language interface file with your application. You can then call the NI-CAN functions without any extra effort.

For C applications (files with `.c` extension), include `nican.h` by adding the following line to the beginning of your code:

```
#include "nican.h"
```

For C++ applications (files with `.cpp` extension), include `nican.h` by adding the following lines to the beginning of your code:

```
#define _cplusplus
#include "nican.h"
```

The `_cplusplus` define allows `nican.h` to properly handle the transition from C++ to the C language NI-CAN functions.

For Microsoft Visual C++ (5.0 or later), link your application with the NI-CAN language interface for Microsoft C/C++, `nicanmsc.lib`.

For Borland C/C++ (5.0 or later), link your application with the NI-CAN language interface for Borland C/C++, `nicanbor.lib`. For Borland C/C++ 4.5, you must use direct entry access for NI-CAN.

For LabWindows/CVI, your application is linked with the NI-CAN language interface for LabWindows/CVI, `nican.lib`. This library is installed automatically based on the installed compatible compiler.

For detailed information on how to compile and link your NI-CAN application, refer to the `readme.txt` file in the NI-CAN `examples` directory.

Direct Entry Access

You can directly access nican.dll from any programming environment that allows you to request addresses of functions that a DLL exports.

To use direct entry access, you must first load nican.dll. The following C language code fragment illustrates how to call the Win32 LoadLibrary function and check for an error:

```
#include <windows.h>
#include "nican.h"

HINSTANCE NicanLib = NULL;

NicanLib=LoadLibrary("nican.dll");
if (NicanLib == NULL) {
   return FALSE;
}
```

Next, your application must use the Win32 GetProcAddress function to get the addresses of the NI-CAN functions your application needs to use. For each NI-CAN function used by your application, you must define a direct entry prototype. For the prototypes for each function exported by nican.dll, refer to the *NI-CAN Programmer Reference Manual*. The following code fragment illustrates how to get the addresses of the ncOpenObject, ncCloseObject, and ncRead functions:

```
static NCTYPE_STATUS (_NCFUNC_ *PncOpenObject)
    (NCTYPE_STRING ObjName,
    NCTYPE_OBJH_P ObjHandlePtr);
static NCTYPE_STATUS (_NCFUNC_ *PncCloseObject)
    (NCTYPE_OBJH ObjHandle);
static NCTYPE_STATUS (_NCFUNC_ *PncRead)
    (NCTYPE_OBJH ObjHandle, NCTYPE_UINT32 DataSize,
    NCTYPE_ANY_P DataPtr);

PncOpenObject = (NCTYPE_STATUS (_NCFUNC_ *)
    (NCTYPE_STRING, NCTYPE_OBJH_P))
    GetProcAddress(NicanLib, (LPCSTR)"ncOpenObject");
PncCloseObject = (NCTYPE_STATUS (_NCFUNC_ *)

    (NCTYPE_OBJH))
    GetProcAddress(NicanLib, (LPCSTR)"ncCloseObject");
PncRead = (NCTYPE_STATUS (_NCFUNC_ *)
    (NCTYPE_OBJH, NCTYPE_UINT32, NCTYPE_ANY_P))
    GetProcAddress(NicanLib, (LPCSTR)"ncRead");
```

If `GetProcAddress` fails, it returns a NULL pointer. The following code fragment illustrates how to verify that none of the calls to `GetProcAddress` failed:

```
if ((PncOpenObject  == NULL) ||
       (PncCloseObject  == NULL) ||
       (PncRead  == NULL)) {
   FreeLibrary(NicanLib);
   printf("GetProcAddress failed");
}
```

Your application needs to de-reference the pointer to access an NI-CAN function, as illustrated by the following code:

```
NCTYPE_STATUS status;
NCTYPE_OBJH MyObjh;

status = (*PncOpenObject) ("CAN0", &MyObjh);
if (status < 0) {
   printf("ncOpenObject failed");
}
```

Before exiting your application, you need to free `nican.dll` with the following command:

```
FreeLibrary(NicanLib);
```

For more information on direct entry, refer to the Microsoft Win32 SDK (Software Development Kit) online help.

Choosing Which NI-CAN Objects to Use

An application written for NI-CAN communicates on the network by using various objects. Which NI-CAN objects to use depends largely on the needs of your application. The following sections discuss the objects provided by NI-CAN, and reasons why you might use each class of object.

Using CAN Network Interface Objects

The CAN Network Interface Object encapsulates a physical interface to a CAN network, usually a CAN port on an AT, PCI, PCMCIA or PXI interface.

You use the CAN Network Interface Object to read and write complete CAN frames. As a CAN frame arrives from over the network, it can be placed into the read queue of the CAN Network Interface Object. You can retrieve CAN frames from this read queue using the `ncRead` or `ncReadMult` function. For CAN Network Interface Objects, the read

functions provide a timestamp of when the frame was received, the arbitration ID of the frame, the type of frame (data or RTSI event), the data length, and the data bytes. You can also use the CAN Network Interface Object to write CAN frames using the `ncWrite` function.

Some possible uses for the CAN Network Interface Object include the following:

- You can use the read queue to log all CAN frames transferred across the network. This log is useful when you need to view preceding CAN traffic to verify that all CAN devices are functioning properly.
- You can use the write queue to transmit a sequence of CAN frames in quick succession. This is useful for applications in which you need to simulate a specific sequence of CAN frames to verify proper device operation.
- You can read and write CAN frames for access to configuration settings within a device. Because such settings generally are not accessed during normal device operation, a dedicated CAN Object is not appropriate.
- For higher level protocols based on CAN, you can use sequences of write/read transactions to initialize communication with a device. In these protocols, specific sequences of CAN frames often need to be exchanged before you can access the data from a device. In such cases, you can use the CAN Network Interface Object to set up communication, then use CAN Objects for actual data transfer with the device.

In general, you use CAN Network Interface Objects for situations in which you need to transfer arbitrary CAN frames.

Using CAN Objects

When a network frame is transmitted on a CAN based network, it always begins with what is called the arbitration ID. This arbitration ID is primarily used for collision resolution when more than one frame is transmitted simultaneously, but you can also use it as a simple mechanism to identify data. The CAN Object encapsulates a specific CAN arbitration ID and its associated data.

Every CAN Object is always associated with a specific CAN Network Interface Object, used to identify the physical interface on which the CAN Object is located. Your application can use multiple CAN Objects in conjunction with their associated CAN Network Interface Object.

The CAN Object provides high level access to a specific arbitration ID. You can configure each CAN Object for different forms of background access. For example, you can configure a CAN Object to transmit a data frame every 100 milliseconds, or to periodically poll for data by transmitting a remote frame and receiving the data frame response. The arbitration ID, direction of data transfer, data length, and when data transfer occurs (periodic or unsolicited) are all preconfigured for the CAN Object. When you have configured and opened the CAN Object, data transfer is handled in the background using read and write queues. For example, if the CAN Object periodically polls for data, the NI-CAN driver automatically handles the periodic transmission of remote frames, and stores incoming data in the read queue of the CAN Object for later retrieval by the `ncRead` function. For CAN Objects that receive data frames, the `ncRead` function provides a timestamp of when the data frame arrived, and the data bytes of the frame. For CAN Objects that transmit data frames, the `ncWrite` function provides the outgoing data bytes.

Some possible uses for CAN Objects include the following:

- You can configure a CAN Object to periodically transmit a data frame for a specific arbitration ID. The CAN Object transmits the same data bytes repetitively until different data is provided using `ncWrite`. This configuration is useful for simulation of a device that periodically transmits its data, such as simulation of an automotive sensor. This configuration is also useful for devices that expect to periodically receive data for a particular arbitration ID to respond with data using a different arbitration ID, such as a device containing analog inputs and outputs.
- You can configure a CAN Object to watch for unsolicited data frames received for its arbitration ID, then store that data in the CAN Object's read queue. A watchdog timeout is provided to ensure that incoming data is received periodically. This configuration is useful when you want to apply a timeout to data received for a specific arbitration ID and store that data in a dedicated queue. If you do not need to apply a timeout for a given arbitration ID, it is preferable to use the CAN Network Interface Object to receive that data.
- You can configure a CAN Object to periodically poll for data by transmitting a remote frame and receiving the data frame response. This configuration is useful for communication with devices that require a remote frame to transmit their data.

- You can configure a CAN Object to transmit a data frame whenever it receives a remote frame for its arbitration ID. You can use this configuration to simulate a device which responds to remote frames.

In general, you use CAN Objects for data transfer for a specific arbitration ID, especially when that data transfer needs to occur periodically.

Programming Model for NI-CAN Applications

The following steps demonstrate how to use the NI-CAN functions in your application. The steps are shown in Figure 2-1 in flowchart form.

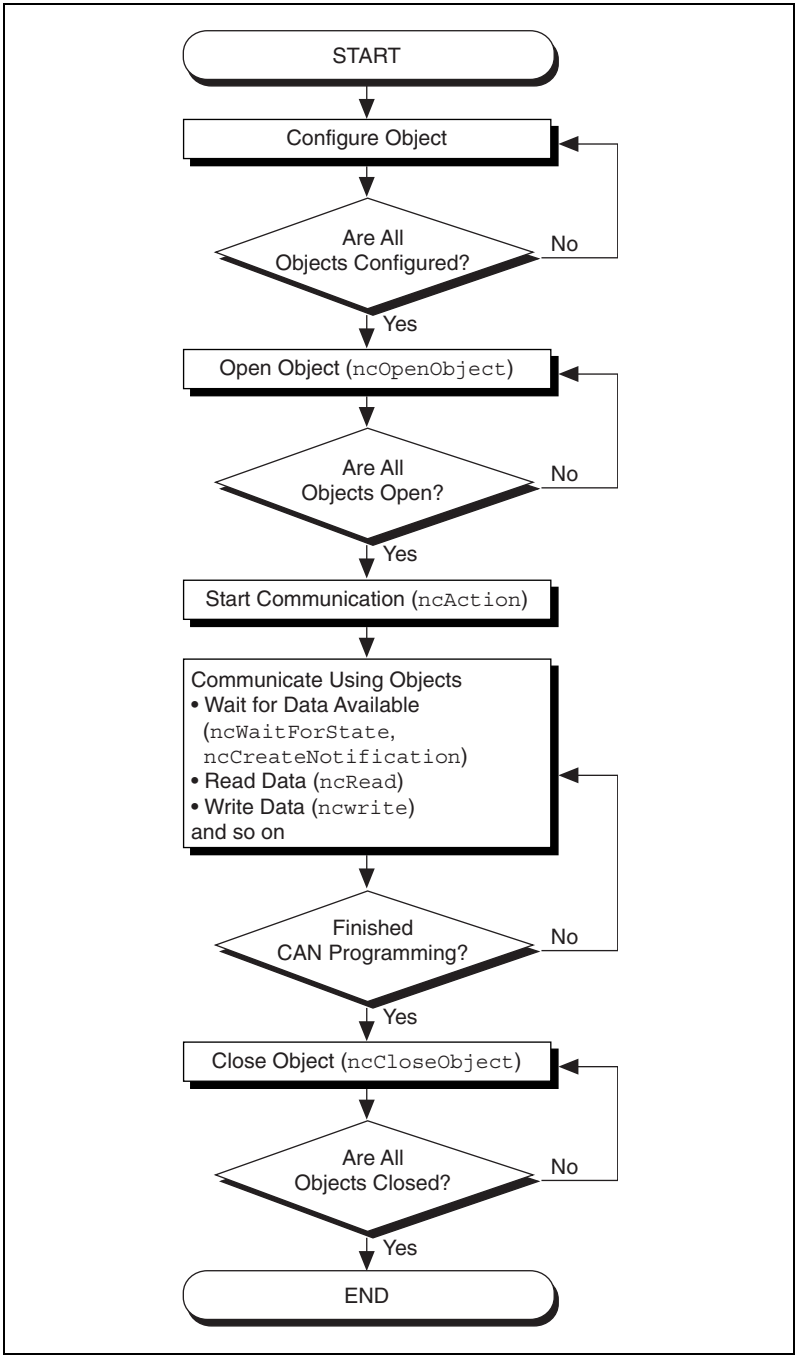


Figure 2-1. General Program Steps Using NI-CAN Functions

Step 1. Configure Objects

Prior to opening the objects used in your application, you must configure the objects with their initial attribute settings. Each object is configured within your application by calling the `ncConfig` function. This function takes the name of the object to configure, along with a list of configuration attribute settings.

Step 2. Open Objects

You must call the `ncOpenObject` function to open each object you use within your application.

The `ncOpenObject` function returns a handle for use in all subsequent NI-CAN calls for that object. When you are using the LabVIEW function library, this handle is passed through the upper left and right terminals of each NI-CAN function used after the open.

Step 3. Start Communication

You must start communication on the CAN network before you can use your objects to transfer data.

If you configured your CAN Network Interface Object to start on open, that object and all of its higher level CAN Objects are started automatically by the `ncOpenObject` function, so nothing special is required for this step.

If you disabled the start-on-open attribute, when your application is ready to start communication, use the CAN Network Interface Object to call the `ncAction` function with the `Opcode` parameter set to `NC_OP_START`. This call is often useful when you want to use `ncWrite` to place outgoing data in write queues prior to starting communication. This call is also useful in high bus load situations, because it is more efficient to start communication after all objects have been opened.

If you want to reset the CAN hardware completely to clear a pending Error Passive state, you can use the CAN Network Interface Object to call the `ncAction` function with the `Opcode` parameter set to `NC_OP_RESET`. This reset must be done prior to starting communication.

Step 4. Communicate Using Objects

After you open your objects and start communication, you are ready to transfer data on the CAN network. The manner in which data is transferred depends on the configuration of the objects you are using. For this example, assume that you are communicating with a CAN device that periodically

transmits a data frame. To receive this data, assume that a CAN Object is configured to watch for data frames received for its arbitration ID and store that data in its read queue.

Step 4a. Wait for Available Data

To wait for the arrival of a data frame from the device, you can call `ncWaitForState` with the `DesiredState` parameter set to `NC_ST_READ_AVAIL`. The `NC_ST_READ_AVAIL` state tells you that data for the CAN Object has been received from the network and placed into the object's read queue. Another way to wait for the `NC_ST_READ_AVAIL` state is to call the `ncCreateNotification` function so you receive a callback when the state occurs. For more information on `ncWaitForState` and `ncCreateNotification`, refer to the *NI-CAN Programmer Reference Manual*.

When receiving data from the device, if your only requirement is to obtain the most recent data, you are not required to wait for the `NC_ST_READ_AVAIL` state. If this is the case, you can set the read queue length of the CAN Object to zero during configuration, so that it only holds the most recent data bytes. Then you can use the `ncRead` function as needed to obtain the most recent data bytes received.

Step 4b. Read Data

Read the data bytes using `ncRead`. For CAN Objects that receive data frames, `ncRead` returns a timestamp of when the data was received, followed by the actual data bytes (the number of which you configured in step 1).

Steps 4a and 4b should be repeated for each data value you want to read from the CAN device.

Step 5. Close Objects

When you are finished accessing the CAN devices, close all objects using the `ncCloseObject` function before you exit your application.

Checking Status of Function Calls

Each NI-CAN function returns a value that indicates the status of the function call. Your application should check this status after each NI-CAN function call. The following sections describe the NI-CAN status, and how you can check it in your application.

Note The NI-CAN status format changed from version 1.4 to version 1.5. If you are developing a new NI-CAN application, this change will not affect your development. If you have an existing NI-CAN application that was developed prior to July 2001, you can either 1.) run a utility to enable backward compatibility for NI-CAN status, or 2.) adapt code to the current NI-CAN status. For more information, refer to the `errors.txt` in the NI-CAN Installation directory.

Checking Status in LabVIEW

For applications written in LabVIEW, status checking is basically handled automatically. For all NI-CAN functions, the lower left and right terminals provide status information using LabVIEW Error Clusters. LabVIEW Error Clusters are designed so that status information flows from one function to the next, and function execution stops when an error occurs. For more information, refer to the *Error Handling* section in the LabVIEW online help.

The LabVIEW Error Clusters returned by NI-CAN functions use the same format as other National Instruments products. You can wire `Error out` from any NI-CAN function to the standard LabVIEW error functions, such as `Simple Error Handler`. The message returned by `Simple Error Handler` will describe the error, including possible solutions.

Table 2-1 summarizes NI-CAN's use of each Error Cluster parameter.

Table 2-1. NI-CAN Error Cluster

Code	Status	Source	Meaning
Negative	True	Name of function where error occurred	Error. Function did not perform expected behavior.
Positive	False	Name of function where warning occurred	Warning. Function performed as expected, but a condition arose that may require your attention.
Zero	False	Empty string	Success. Function completed successfully.

Within your LabVIEW Block Diagram, wire the `Error in` and `Error out` terminals of all NI-CAN functions together in succession. When an error is detected in any NI-CAN function (`status` parameter true), all subsequent NI-CAN functions are skipped except for `ncClose`. The `ncClose` function executes regardless of whether the incoming `status` is true or false. This ensures that all NI-CAN objects are closed properly when execution stops due to an error.

When a warning occurs in an NI-CAN function, execution proceeds normally. To detect suspected warnings in your application, you can write code in your block diagram to examine the `code` parameter, or you can use the `Probe Data` tool on an `Error out` terminal during execution.

Checking Status in C or C++

For applications written in C or C++, each NI-CAN function returns a status code as a signed 32-bit integer. Table 2-2 summarizes NI-CAN's use of this status:

Table 2-2. NI-CAN Status Code

Status Code	Meaning
Negative	Error. Function did not perform expected behavior.
Positive	Warning. Function performed as expected, but a condition arose that may require your attention.
Zero	Success. Function completed successfully.

Your application code should check the status returned from every NI-CAN function. If an error is detected, you should close all NI-CAN handles, then exit the application. If a warning is detected, you can display a message for debugging purposes, or simply ignore the warning.

To assist with debugging, NI-CAN provides a function you can use to display a message that describes the error/warning, including possible solutions. This `ncStatusToString` function takes a status code as input, then returns a descriptive string. For more information on `ncStatusToString`, refer to the *NI-CAN Programmer Reference Manual*.

The following piece of code shows an example of handling NI-CAN status during application debugging:

```
status= ncOpenObject ("CAN0", &MyObjHandle);
PrintStat (status, "ncOpen CAN0");
```

where the function `PrintStat` has been defined at the top of the program as:

```
void PrintStat(NCTYPE_STATUS status, char *source)
{
    char statusString[512];
    if(status != 0)
    {
        ncStatusToString(status, sizeof(statusString),
                         StatusString);
        printf("\n%s\nSource = %s\n", statusString,
                source);
        if (status < 0)
        {
            ncCloseObject(MyObjHandle);
            exit(1);
        }
    }
}
```

In some situations, you may want to check for specific errors in your code. For example, when `ncWaitForState` times out, you may want to continue communication, rather than exit the application. To check for specific errors, use the constants defined in `nican.h`. These constants have the same names as described in the *NI-CAN Programmer Reference Manual*. For example, to check for a function timeout:

```
if (status == CanErrFunctionTimeout)
```

3

NI-CAN Programming Techniques

This chapter describes techniques for using the NI-CAN functions in your application.

For more detailed information about each NI-CAN function, refer to the *NI-CAN Programmer Reference Manual*.

Using Queues

To maintain an ordered history of data transfers, NI-CAN supports the use of queues, also known as FIFO (first-in-first-out) buffers. The basic behavior of such queues is common to all NI-CAN objects.

There are two basic types of NI-CAN queues: the read queue and the write queue. NI-CAN uses the read queue to store incoming network data items in the order they arrive. You access the read queue using `ncRead` to obtain the data. NI-CAN uses the write queue to transmit network frames one at a time using the network interface hardware. You access the write queue using `ncWrite` to store network data items for transmission.

State Transitions

The `NC_ST_READ_AVAIL` state transitions from false to true when NI-CAN places a new data item into an empty read queue, and remains true until you read the last data item from the queue and the queue is empty.

The `NC_ST_READ_MULT` state transitions from false to true when the number of items in a queue exceeds a threshold. The threshold is set using the `NC_ATTR_NOTIFY_MULT_LEN` attribute. The `NC_ST_READ_MULT` state and `ncReadMult` function are useful in high-traffic networks in which data items arrive quickly.

The `NC_ST_WRITE_SUCCESS` state transitions from false to true when the write queue is empty and NI-CAN has successfully transmitted the last data item onto the network. The `NC_ST_WRITE_SUCCESS` state remains true until you write another data item into the write queue.

Empty Queues

For both read and write queues, the behavior for reading an empty queue is similar. When you read an empty queue, the previous data item is returned again. For example, if you call `ncRead` when `NC_ST_READ_AVAIL` is false, the data from the previous call to `ncRead` is returned again, along with the `CanWarnOldData` warning. If no data item has yet arrived for the read queue, a default data item is returned, which consists of all zeros. You should generally wait for `NC_ST_READ_AVAIL` prior to the first call to `ncRead`.

Full Queues

For both read and write queues, the behavior for writing a full queue is similar. When you write a full queue, NI-CAN returns the `CanErrOverflowWrite` error codes. For example, if you write too many data items to a write queue, the `ncWrite` function eventually returns the overflow error.

Disabling Queues

If you do not need a complete history of all data items, you can disable the read queue and/or write queue by setting its length to zero. Zero length queues are typically used only with CAN objects, not the CAN Network Interface Object. Using zero length queues generally saves memory, and often results in better performance. When a new data item arrives for a zero length queue, it overwrites the previous item without indicating an overflow. The `NC_ST_READ_AVAIL` and `NC_ST_WRITE_SUCCESS` states still behave as usual, but you can ignore them if you want only the most recent data. For example, when NI-CAN writes a new data item to the read buffer, the `NC_ST_READ_AVAIL` state becomes true until the data item is read. If you only want the most recent data, you can ignore the `NC_ST_READ_AVAIL` state, as well as the `CanWarnOldData` warning returned by `ncRead`.

Using the CAN Network Interface Object with CAN Objects

For many applications, it is desirable to use a CAN Network Interface Object in conjunction with higher level CAN Objects. For example, you can use CAN objects to transmit data or remote frames periodically, and use the CAN Network Interface Object to receive all incoming frames. For more information on the different uses of NI-CAN objects, refer to the

Choosing Which NI-CAN Objects to Use section in Chapter 2, *Developing Your Application*.

When one or more CAN Objects are open, the CAN Network Interface Object cannot receive frames which would normally be handled by the CAN Objects. The flowchart in Figure 3-1 shows the steps performed by NI-CAN when a CAN frame is received.

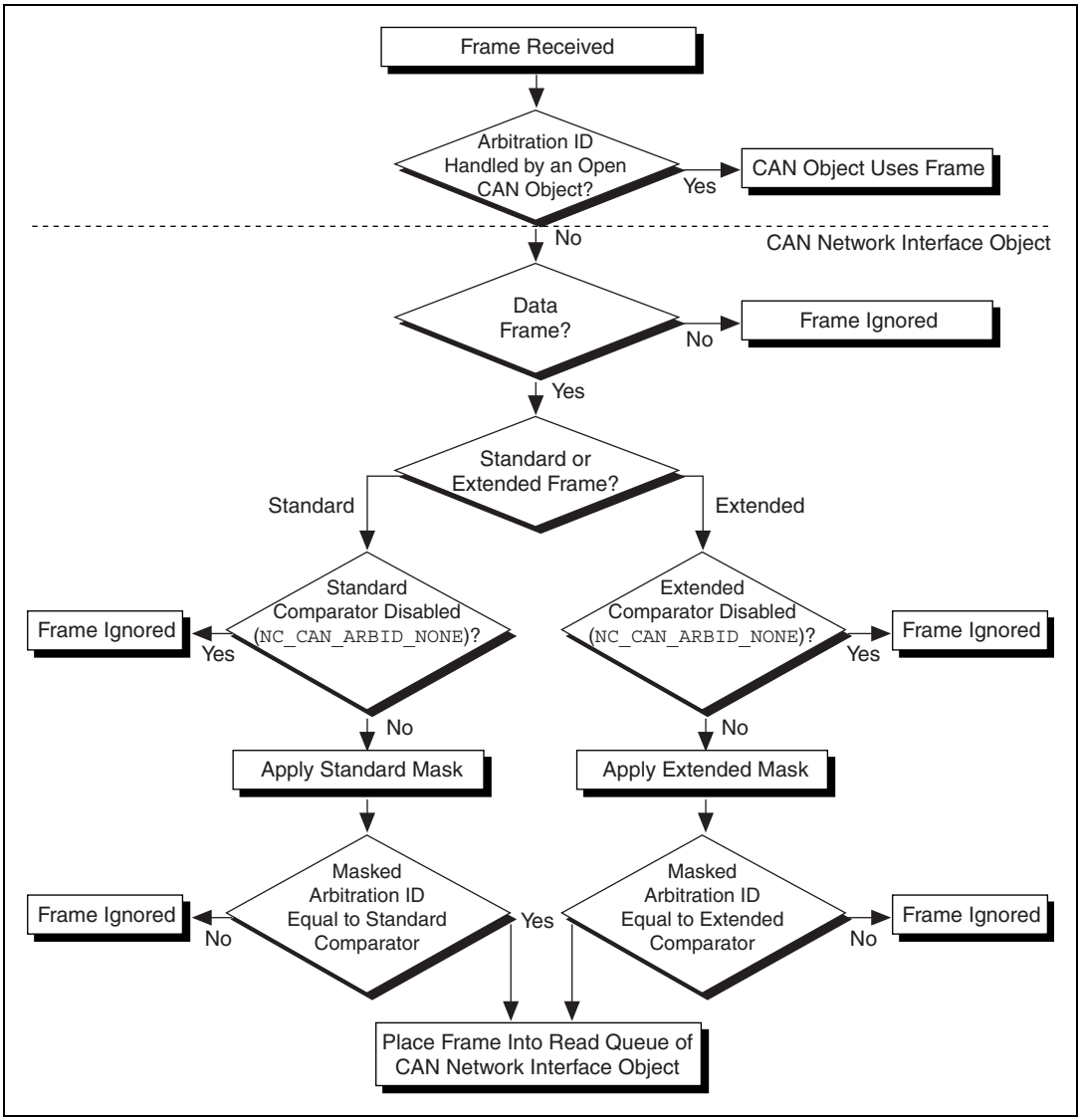


Figure 3-1. Flowchart for CAN Frame Reception

The decisions in Figure 3-1 are generally performed by the on-board CAN communications controller chip. Nevertheless, if you intend to use CAN Objects as the sole means of accessing the CAN bus, it is best to disable all frame reception in the CAN Network Interface Object by setting the comparator attributes to NC_CAN_ARBID_NONE (hex CFFFFFFF). By doing this, the CAN communications controller chip is best able to filter out all incoming frames except those handled by CAN Objects.

Detecting State Changes

You can detect state changes for an object using one of the following schemes:

- Call ncWaitForState to wait for one or more states to occur.
- Use ncCreateNotification in C/C++ to register a callback for one or more states.
- Use ncCreateOccurrence in LabVIEW to create an occurrence for one or more states.
- Call ncGetAttribute to get the NC_ATTR_STATE attribute.

Use the ncWaitForState function when your application must wait for a specific state before proceeding. For example, if you call ncWrite to write a frame, and your application cannot proceed until the frame is successfully transmitted, you can call ncWaitForState to wait for NC_ST_WRITE_SUCCESS.

Use the ncCreateNotification function in C/C++ when your application must handle a specific state, but can perform other processing while waiting for that state to occur. The ncCreateNotification function registers a callback function, which is invoked when the desired state occurs. For example, a callback function for NC_ST_READ_AVAIL can call ncRead and place the resulting data in a buffer. Your application can then perform any tasks desired, and process the CAN data only as needed.

Use the ncCreateOccurrence function in LabVIEW when your application must handle a specific state, but can perform other processing while waiting for that state to occur. The ncCreateOccurrence function creates a LabVIEW occurrence, which is set when the desired state occurs. Occurrences are the mechanism used in LabVIEW to provide multithreaded execution.

Use the ncGetAttribute function when you need to determine the current state of an object.

4

Application Examples

This chapter describes how to explore the sample applications provided with your NI-CAN software.

LabVIEW Examples

The NI-CAN examples for LabVIEW are located in the `LabVIEW\examples\nican` folder.

For LabVIEW 6.0, you can access the NI-CAN example information at **Help»Examples»Other NI Products»Controller Area Network (CAN)**. Other versions of LabVIEW include similar help.

The help describes each example and includes a link you can use to open the VI.

C/C++ Examples

The NI-CAN examples for C/C++ are in the `examples` folder of your NI-CAN directory. The default path is `C:\Program Files\National Instruments\NI-CAN\examples`. These examples are also in the `CVI\samples\nican` folder if you have CVI on your system.

The `readme.txt` file in the `examples` folder describes each example, including the names of the associated files. You can build the examples using LabWindows/CVI, Microsoft Visual C/C++, or Borland C/C++.

Other Programming Languages

Although the NI-CAN software does not include examples for other programming languages, you may find assistance on the National Instruments Web site, `ni.com`. For more information, see Appendix I, *Technical Support Resources*, in this manual.

5

NI-CAN Configuration and Diagnostic Utilities

This chapter describes the NI-CAN Configuration and Diagnostic utilities you can use to configure and diagnose the objects of the NI-CAN software.

Overview

The Windows Me/98/95 NI-CAN Configuration utility is integrated into the Windows Device Manager. The Windows 2000/NT NI-CAN Configuration utility is integrated into the Windows 2000/NT Control Panel. For each CAN interface in your system, you can use the NI-CAN Configuration utility to configure each CAN port as a CAN Network Interface Object. For example, you can configure the two ports of a PCI-CAN/2 as `CAN0` and `CAN1`.

You can use the NI-CAN Diagnostic utility, installed with your NI-CAN software, to test the hardware and software installation. The utility verifies that your hardware and software are functioning properly and that the configuration of your CAN interfaces does not conflict with anything else in your system.

Starting the NI-CAN Configuration Utility in Windows Me/98/95

To start the NI-CAN Configuration utility on Windows Me/98/95, follow these steps.

1. Double-click on the **System** icon in the **Control Panel**, which you can open from the **Settings** selection of the **Start** menu.
2. Select the **Device Manager** tab in the **System Properties** dialog box that appears.
3. Click on the **View devices by type** button at the top of the **Device Manager** tab, and double-click on **National Instruments CAN Interfaces**.

4. In the list of installed interfaces immediately below **National Instruments CAN Interfaces**, double-click on the particular interface type you want to configure. If you have only one National Instruments CAN interface in your computer, only one interface type appears in the list. If an exclamation point or an X appears next to the interface, there is a problem, and you should refer to the *Problem Shown in Device Manager* section of Appendix A, *Windows Me/98/95: Troubleshooting and Common Questions*, in this manual, to resolve your problem before you continue. Use the **Resources** tab to provide information about the hardware resources assigned to the CAN interface, and use the **Settings** tab to assign a name to each CAN port.

Starting the NI-CAN Configuration Utility in Windows 2000/NT

To start the NI-CAN Configuration utility on Windows 2000/NT, open your Windows 2000/NT Control Panel and double-click on **NI-CAN Configuration**.

Because you can use the NI-CAN Configuration utility to modify the configuration of the NI-CAN kernel drivers, you must be logged on to Windows 2000/NT as the `Administrator` to make any changes. If you start the NI-CAN Configuration utility without `Administrator` privileges, it runs in read-only mode; you can view the settings, but you cannot make changes.

The main dialog box of the NI-CAN Configuration utility for Windows 2000/NT contains a list of all National Instruments CAN interfaces in your computer. For each CAN interface, the **Resources** button opens a dialog box you can use to specify hardware resources for the CAN interface, and the **Settings** button opens a dialog box you can use to assign a name to each CAN port. Windows 2000 is fully Plug and Play, so the resources are read-only. Therefore the **Resources** button is disabled in Windows 2000.

After you have finished configuring your CAN interfaces, click on the **OK** button to close the dialog box.

Starting the NI-CAN Remote Configuration Utility for LabVIEW RT

To start the NI-CAN Remote Configuration Utility, select **Start»National Instruments»NI-CAN»NI-CAN Remote Configuration**.

Use the NI-CAN Remote Configuration Utility to configure and diagnose CAN interfaces in a remote PXI chassis running LabVIEW RT. You run the utility on your Windows machine, and it communicates over Ethernet to configure the remote PXI chassis.

When you start the NI-CAN Remote Utility, the first dialog box prompts you for the IP address (non-DHCP) or machine name (DHCP) of the remote PXI chassis. If you do not know this information, you can select the **Launch MAX** button to start Measurement and Automation Explorer (MAX). You can use **Remote Systems** in MAX to find a remote machine's IP address and name.

After selecting **OK** in the initial dialog box, use the main dialog to configure and diagnose CAN interfaces in a manner similar to the Windows utilities. Use the **Test All** and **Text One** buttons to diagnose proper installation and operation.

If you want to change the NI-CAN interface name associated with a port, double-click on the port name. The resulting dialog box provides an **Interface Name** drop-down box you can use to select **CAN0**, **CAN1**, and so on. Changing the NI-CAN interface name is normally done only when you have multiple cards in your system.

After you finish configuring your CAN interfaces, click on the **Exit** button to close the utility.

Starting the NI-CAN Diagnostic Utility

To run the utility, select the **NI-CAN Diagnostic** item under **Start»Programs»National Instruments CAN»NI-CAN**.

When you have started the NI-CAN Diagnostic utility, test your CAN interfaces by clicking on the **Test All** button. You can also test one CAN interface by highlighting it and clicking on the **Test One** button. If the NI-CAN Diagnostic is successful, it puts a checkmark next to the interface and changes its status from "Untested" to "Passed." If the NI-CAN Diagnostic fails, it puts an X next to the interface, and changes its status

from “Untested” to “Failed.” Figure 5-1 shows the NI-CAN Diagnostic utility after it has tested some CAN interfaces.

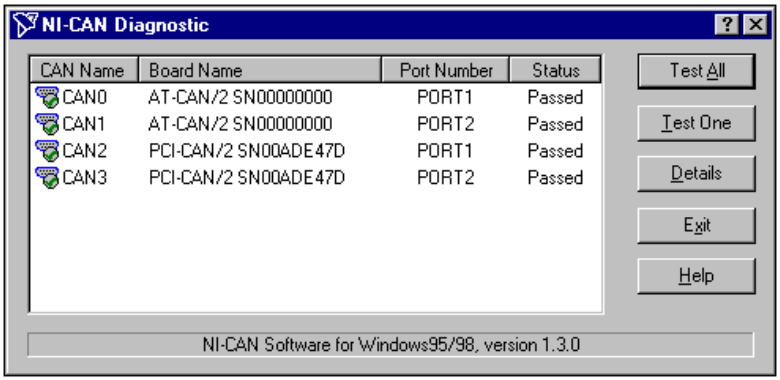


Figure 5-1. NI-CAN Diagnostic Utility after Testing

You can get details about any tested CAN interface by selecting the interface and clicking the **Details** button. For each failed CAN interface, select it and click the **Details** button to get a description of the failure. Use that information and the information in Appendix A, *Windows Me/98/95: Troubleshooting and Common Questions*, Appendix B, *Windows NT: Troubleshooting and Common Questions*, or Appendix C, *Windows 2000: Troubleshooting and Common Questions*, to troubleshoot the problem. Troubleshooting information is also available in the online help for the NI-CAN Diagnostic utility, which you can access by clicking on the **Help** button.

A

Windows Me/98/95: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems with the NI-CAN software for Windows Me/98/95 and answers some common questions.

Troubleshooting Windows Device Manager Problems

The Windows Device Manager contains configuration information for all of the CAN hardware it is aware of that is installed in your system. To start the Windows Device Manager, double-click on the **System** icon under **Start»Settings»Control Panel**. In the **System Properties** box that appears, select the **Device Manager** tab and click the **View devices by type** radio button at the top of the tab.

If there is no **National Instruments CAN Interfaces** item, and you are certain you have a CAN interface installed, refer to the *No National Instruments CAN Interfaces* section of this appendix.

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, refer to the *Missing CAN Interface* section of this appendix.

If the CAN interface you are looking for is listed, but has a circled X or exclamation mark (!) over its icon, refer to the *Problem Shown in Device Manager* section of this appendix.

No National Instruments CAN Interfaces

If you are certain you have a Plug and Play CAN interface installed, but no **National Instruments CAN Interfaces** item appears in the **Device Manager**, the interface is probably incorrectly listed under **Other Devices**. Double-click on the **Other Devices** item in the Device Manager and, one by one, remove each National Instruments CAN interface listed there by selecting its name and then clicking the **Remove** button. After all of the

National Instruments CAN interfaces have been removed from **Other Devices**, click the **Refresh** button. At this point, the system rescans the installed hardware, and the CAN interface should show up under **National Instruments CAN Interfaces** without any problems. If the problem persists, contact National Instruments.

Missing CAN Interface

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, the CAN interface is not properly installed. For National Instruments CAN hardware, this problem indicates that the interface is not physically present in the system.

Problem Shown in Device Manager

If a CAN interface is not working properly, its icon has a circled X or exclamation mark (!) overlaid on it, as shown in Figure A-1.

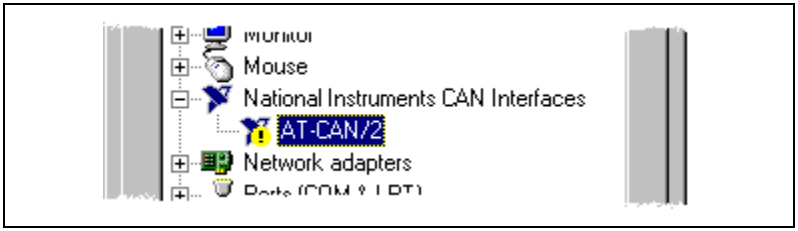


Figure A-1. CAN Interface That Is Not Working Properly

This problem can occur for several reasons. If you encounter this problem, the Device Manager should list an error code that indicates why the problem occurred. To see the error code for a particular interface, select the name of the interface and click on the **Properties** button to go to the **General** tab for that CAN interface. The **Device Status** section of the **General** tab shows the error code. Locate the error code in the following list to find out why your CAN interface is not working properly:

- Code 8—The NI-CAN software was incompletely installed. To solve this problem, reinstall the NI-CAN software.
- Code 9—Windows Me/98/95 had a problem reading information from the CAN interface. Contact National Instruments for assistance.
- Code 12—The CAN interface was not assigned a physical memory range. If your computer does not have 8 KB of available memory, Windows Me/98/95 might configure your CAN interface without a physical memory assignment. The NI-CAN software cannot function without 8 KB of physical memory. Another way to verify this problem

is to look at the **Resource settings** list on the **Resources** tab to verify that the CAN interface was not assigned a Memory Range. To solve this problem, free up an 8 KB Memory Range (such as D0000 to D1FFF hex) that is being used by another device in the system.

- Code 15—The CAN interface was not assigned an Interrupt Request level. If your computer does not have any available Interrupt Request levels, Windows Me/98/95 might configure your CAN interface without an Interrupt Request level. The NI-CAN software cannot function without an Interrupt Request level. Another way to verify this problem is to look at the **Resource settings** list on the **Resources** tab to verify that the CAN interface was not assigned an Interrupt Request level. To solve this problem, free up an Interrupt Request level that is being used by another device in the system.
- Code 22—The CAN interface is disabled. To enable the CAN interface, check the appropriate configuration checkbox in the **Device Usage** section of the **General** tab.
- Code 24—The CAN interface is not present, or the Device Manager is unaware that the CAN interface is present. To solve this problem, select the interface in the Device Manager, and click on the **Remove** button. Next, click the **Refresh** button. At this point, the system rescans the installed hardware, and the CAN interface should show up without any problems. If the problem persists, contact National Instruments.
- Code 27—Windows Me/98/95 was unable to assign the CAN interface any resources. To solve this problem, free up system resources by disabling other unnecessary hardware so that enough resources are available for the CAN interface. The resources required for a single CAN interface are an Interrupt Request level and an 8 KB physical Memory Range (such as D0000 to D1FFF hex).

Troubleshooting Diagnostic Utility Failures

The following sections explain common error messages generated by the NI-CAN Diagnostic utility.

Memory Resource Conflict

This error occurs if the memory resource assigned to a CAN interface conflicts with the memory resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use resources that have not been reserved properly with the Device Manager. If a resource conflict exists, write down the memory

resource that caused the conflict and refer to the *Microsoft Windows User's Guide* for instructions on how to use the Device Manager to reserve memory resources for legacy boards. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

Interrupt Resource Conflict

This error occurs if the interrupt resource assigned to a CAN interface conflicts with the interrupt resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use resources that have not been reserved properly with the Device Manager. If a resource conflict exists, write down the interrupt resource that caused the conflict and refer to the *Microsoft Windows User's Guide* for instructions on how to use the Device Manager to reserve interrupt resources for legacy boards. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

NI-CAN Software Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects that it is unable to communicate correctly with the CAN hardware using the installed NI-CAN software. If you get this error, shut down your computer, restart it, and run the NI-CAN Diagnostic utility again. If the problem persists, try reinstalling the NI-CAN software.

Missing CAN Interface

If a National Instruments CAN interface is physically installed in your system, but is not listed in the NI-CAN Diagnostic utility, check the Windows Device Manager to see if Windows Me/98/95 has detected the hardware. For more information, refer to the *Troubleshooting Windows Device Manager Problems* section, earlier in this appendix.

CAN Hardware Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects a defect in the CAN hardware. If you get this error, write down the numeric code shown with the error, and contact National Instruments. Depending on the cause of the hardware failure, National Instruments may need to upgrade your CAN interface.

Common Questions

What do I do if my CAN hardware is listed in the Windows Device Manager with a circled X or exclamation point (!) overlaid on it?

Refer to the *Problem Shown in Device Manager* section of this appendix for specific information about what might cause this problem. If you have already completed the troubleshooting steps, contact National Instruments.

How can I determine which type of CAN hardware I have installed?

Run the NI-CAN Configuration utility. To run the utility, select **Start»Settings»Control Panel»System**. Select the **Device Manager** tab in the **System Properties** dialog box. Click on the **View devices by type** radio button at the top of the sheet. If any CAN hardware is correctly installed, a **National Instruments CAN Interfaces** icon appears in the list of device types. Double-click this icon to see a list of installed CAN hardware.

How can I determine which version of the NI-CAN software I have installed?

Run the NI-CAN Diagnostic utility. To run the utility, select **NI-CAN Diagnostics** under **Start»Programs»National Instruments»NI-CAN**. The NI-CAN Diagnostic utility displays information about the version of the NI-CAN software currently installed.

What do I do if the NI-CAN Diagnostic utility fails with an error?

Refer to the *Troubleshooting Diagnostic Utility Failures* section of this appendix for specific information about what might cause the NI-CAN Diagnostic utility to fail. If you have already completed the troubleshooting steps, contact National Instruments.

How many CAN interfaces can I configure for use with my NI-CAN software for Windows Me/98/95?

The NI-CAN software for Windows Me/98/95 can be configured to communicate with up to 10 CAN interfaces.

Are interrupts required for the NI-CAN software for Windows Me/98/95?

Yes, one interrupt per interface is required.

How do I use an NI-CAN language interface?

For information about using NI-CAN language interfaces, refer to Chapter 2, *Developing Your Application*.

How do I use NI-CAN from within LabVIEW?

For information about using NI-CAN from within LabVIEW, refer to Chapter 2, *Developing Your Application*.

Why does the uninstall program leave some components installed?

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory, because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

B

Windows NT: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems with the NI-CAN software for Windows NT and answers some common questions.

Missing CAN Interface in the NI-CAN Configuration Utility

The NI-CAN Configuration utility contains configuration information for all of the CAN hardware it is aware of that is installed in your system. To start the NI-CAN Configuration utility, double-click on **NI-CAN Configuration** under **Start»Settings»Control Panel**.

If the CAN interface you are looking for is not listed under **National Instruments CAN Interfaces**, the CAN interface is not properly installed. For National Instruments CAN hardware, this problem indicates that the interface is not physically present in the system. If the interface is firmly plugged into its slot and the problem persists, contact National Instruments.

Troubleshooting Diagnostic Utility Failures

The following sections explain common error messages generated by the NI-CAN Diagnostic utility.

No Resources Assigned

This error occurs if you have not assigned resources to the CAN interface. Use the **Resources** button of the NI-CAN Configuration utility to select valid resources for your CAN interface.

Memory Resource Conflict

This error occurs if the memory resource assigned to a CAN interface conflicts with the memory resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use the resources you assigned using the NI-CAN Configuration utility. If a resource conflict exists, use the **Resources** button in the NI-CAN Configuration utility to assign a different memory resource to the CAN interface. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

Interrupt Resource Conflict

This error occurs if the interrupt resource assigned to a CAN interface conflicts with the interrupt resources being used by other devices in the system. Resource conflicts typically occur when your system contains legacy boards that use the resources you assigned using the NI-CAN Configuration utility. If a resource conflict exists, use the **Resources** button in the NI-CAN Configuration utility to assign a different interrupt resource to the CAN interface. After the conflict has been resolved, run the NI-CAN Diagnostic utility again.

NI-CAN Software Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects that it is unable to communicate correctly with the CAN hardware using the installed NI-CAN software. If you get this error, shut down your computer, restart it, and run the NI-CAN Diagnostic utility again. If the problem persists, try reinstalling the NI-CAN software.

Missing CAN Interface

If a National Instruments CAN interface is physically installed in your system, but is not listed in the NI-CAN Diagnostic utility, check to see if the NI-CAN Configuration utility has detected the hardware. For more information, refer to the *Missing CAN Interface in the NI-CAN Configuration Utility* section earlier in this appendix.

CAN Hardware Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects a defect in the CAN hardware. If you get this error, write down the numeric code shown with the error and contact National Instruments. Depending on the cause of the hardware failure, National Instruments may need to upgrade your CAN interface.

Common Questions

How can I determine which type of CAN hardware I have installed?

Run the NI-CAN Configuration utility. To run the utility, select **Start»Settings»Control Panel»NI-CAN Configuration**. If any CAN hardware is correctly installed, it is listed under **National Instruments CAN Interfaces**.

How can I determine which version of the NI-CAN software I have installed?

Run the NI-CAN Diagnostic utility. To run the utility, select **NI-CAN Diagnostics** under **Start»Programs»National Instruments»NI-CAN**. The NI-CAN Diagnostic utility displays information about the version of the NI-CAN software currently installed.

What do I do if the NI-CAN Diagnostic utility fails with an error?

Refer to the *Troubleshooting Diagnostic Utility Failures* section of this appendix for specific information about what might cause the NI-CAN Diagnostic utility to fail. If you have already completed the troubleshooting steps, contact National Instruments.

How many CAN interfaces can I configure for use with my NI-CAN software for Windows NT?

The NI-CAN software can be configured to communicate with up to 10 CAN interfaces.

Are interrupts required for the NI-CAN software for Windows NT?

Yes, one interrupt per card is required.

How do I use an NI-CAN language interface?

For information about using NI-CAN language interfaces, refer to Chapter 2, *Developing Your Application*.

How do I use NI-CAN from within LabVIEW?

For information about using NI-CAN from within LabVIEW, refer to Chapter 2, *Developing Your Application*.

Why does the uninstall program leave some components installed?

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

C

Windows 2000: Troubleshooting and Common Questions

This appendix describes how to troubleshoot problems with the NI-CAN software for Windows 2000 and answers some common questions.

Troubleshooting Windows Device Manager Problems

The Windows Device Manager contains configuration information for all of the CAN hardware it is aware of that is installed in your system. To start the Windows Device Manager, double-click on the **System** icon under **Start»Settings»Control Panel**. In the **System Properties** box that appears, select the **Hardware** tab and click the **Device Manager** button.

If there is no **National Instruments CAN Interfaces** item and you are certain you have a CAN interface installed, refer to the *No National Instruments CAN Interfaces* section of this appendix.

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, refer to the *Missing CAN Interface* section of this appendix.

If the CAN interface you are looking for is listed, but has a circled X or exclamation mark (!) over its icon, or if it appears under **Other Devices** as a **PCI Simple Communication Controller**, refer to the *Problem Shown in Device Manager* section of this appendix.

No National Instruments CAN Interfaces

If you are certain you have a Plug and Play CAN interface installed, but no **National Instruments CAN Interfaces** item appears in the **Device Manager**, the interface is probably incorrectly listed under **Other Devices**. Double-click on the **Other Devices** item in the Device Manager and, one by one, remove each National Instruments CAN interface listed there by selecting its name and then clicking the **Uninstall** button. After all of the

National Instruments CAN interfaces have been removed from **Other Devices**, click the **Scan for Hardware Changes** button. At this point, the system rescans the installed hardware, and the CAN interface should show up under **National Instruments CAN Interfaces** without any problems. If the problem persists, contact National Instruments.

Missing CAN Interface

If the **National Instruments CAN Interfaces** item exists, but the CAN interface you are looking for is not listed there, the CAN interface is not properly installed. For National Instruments CAN hardware, this problem indicates that the interface is not physically present in the system.

Problem Shown in Device Manager

If a CAN interface is not working properly, its icon has a circled X or exclamation mark (!) overlaid on it, as shown in Figure C-1.

Figure C-1. CAN Interface That Is Not Working Properly

This problem can occur for several reasons. If you encounter this problem, the **Device Manager** should list troubleshooting information. To launch the troubleshooter for a particular interface, select the name of the interface and click on the **Properties** button to go to the **General** tab for that CAN interface. Click on the **Troubleshooter...** button to diagnose and solve the problem.

If a CAN interface has not been properly recognized by the NI-CAN driver software, it will appear in the **Device Manager** under **Other Devices** as a **PCI Simple Communication Controller** as shown in Figure C-2.

Figure C-2. CAN Interface That has Not Been Recognized Properly

This problem is likely to occur if you upgraded from a previous version of NI-CAN that did not support Windows 2000 without uninstalling the CAN hardware. To fix this problem, select the device and click the **Uninstall** button, then **Scan for Hardware Changes**. Windows 2000 should identify the device as a National Instruments CAN interface.

Troubleshooting Diagnostic Utility Failures

The following sections explain common error messages generated by the NI-CAN Diagnostic utility.

NI-CAN Software Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects that it is unable to communicate correctly with the CAN hardware using the installed NI-CAN software. If you get this error, shut down your computer, restart it, and run the NI-CAN Diagnostic utility again. If the problem persists, try reinstalling the NI-CAN software.

Missing CAN Interface

If a National Instruments CAN interface is physically installed in your system, but is not listed in the NI-CAN Diagnostic utility, check the Windows Device Manager to see if Windows 2000 has detected the hardware. For more information, refer to the *Troubleshooting Windows Device Manager Problems* section in this appendix.

CAN Hardware Problem Encountered

This error occurs if the NI-CAN Diagnostic utility detects a defect in the CAN hardware. If you get this error, write down the numeric code shown with the error, and contact National Instruments. Depending on the cause of the hardware failure, National Instruments may need to upgrade your CAN interface.

Common Questions

What do I do if my CAN hardware is listed in the Windows Device Manager with a circled X or exclamation point (!) overlaid on it, or if CAN hardware is listed under Other Devices?

Refer to the *Problem Shown in Device Manager* section of this appendix for specific information about what might cause this problem. If you have already completed the troubleshooting steps, contact National Instruments.

How can I determine which type of CAN hardware I have installed?

Run the NI-CAN Configuration utility. To run the utility, select **Start»Settings»Control Panel»NI-CAN Configuration**. If any CAN hardware is correctly installed, it is listed under **National Instruments CAN Interfaces**.

How can I determine which version of the NI-CAN software I have installed?

Run the NI-CAN Diagnostic utility. To run the utility, select **NI-CAN Diagnostics** under **Start»Programs»National Instruments»NI-CAN**. The NI-CAN Diagnostic utility displays information about the version of the NI-CAN software currently installed.

What do I do if the NI-CAN Diagnostic utility fails with an error?

Refer to the *Troubleshooting Diagnostic Utility Failures* section of this appendix for specific information about what might cause the NI-CAN Diagnostic utility to fail. If you have already completed the troubleshooting steps, contact National Instruments.

How many CAN interfaces can I configure for use with my NI-CAN software for Windows 2000?

The NI-CAN software can be configured to communicate with up to 10 CAN interfaces.

Are interrupts required for the NI-CAN software for Windows 2000?

Yes, one interrupt per card is required.

How do I use an NI-CAN language interface?

For information about using NI-CAN language interfaces, refer to Chapter 2, *Developing Your Application*.

How do I use NI-CAN from within LabVIEW?

For information about using NI-CAN from within LabVIEW, refer to Chapter 2, *Developing Your Application*.

Why does the uninstall program leave some components installed?

The uninstall program removes only items that the installation program installed. If you add anything to a directory that was created by the installation program, the uninstall program does not delete that directory because the directory is not empty after the uninstallation. You will need to remove any remaining components yourself.

D

Cabling Requirements for High-Speed CAN

This section describes the cabling requirements for high-speed CAN hardware.

Cables should be constructed to meet these requirements, as well as the requirements of the other CAN or DeviceNet devices in the network.

Connector Pinouts

Depending on the type of CAN interface you are installing, the CAN hardware has DB-9 D-Sub connectors(s), or Combicon-style pluggable screw terminal connector(s), or both.

The 9-pin D-Sub follows the pinout recommended by CiA DS 102. Figure D-1 shows the pinout for this connector.

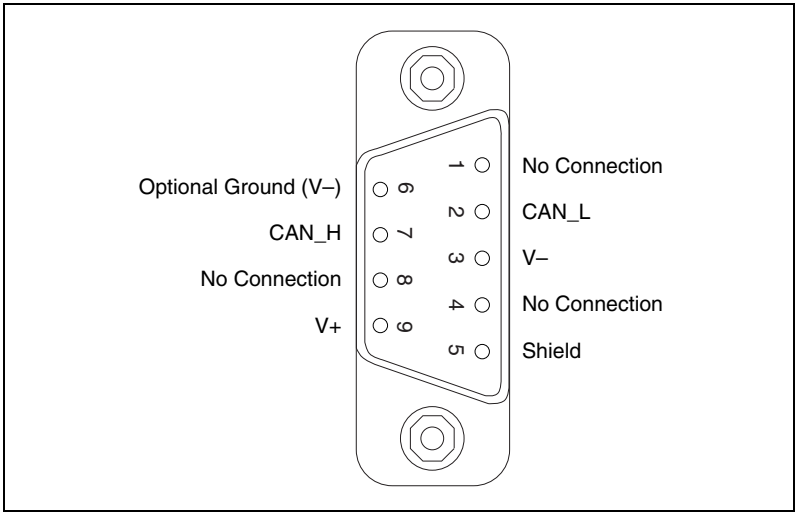


Figure D-1. Pinout for 9-Pin D-Sub Connector

The 5-pin Combicon-style pluggable screw terminal follows the pinout required by the DeviceNet specification. Figure D-2 shows the pinout for this connector.

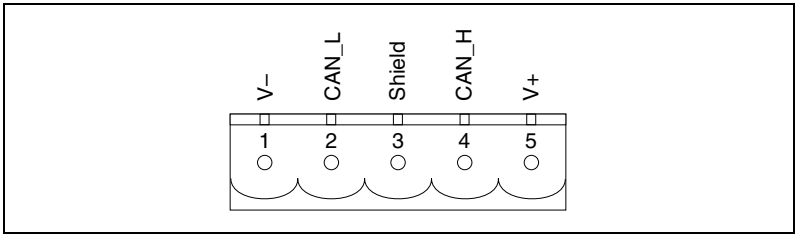


Figure D-2. Pinout for 5-Pin Combicon-Style Pluggable Screw Terminal

CAN_H and CAN_L are signal lines that carry the data on the CAN network. These signals should be connected using twisted-pair cable.

The V+ and V– pins are used to supply bus power to the CAN physical layer if external power is required for the CAN physical layer. If internal power for the CAN physical layer is used, the V- pin serves as the reference ground for CAN_H and CAN_L. See the next section, *Power Supply Information for the High-Speed CAN Ports*, for more information.

Figure D-3 shows the end of a PCMCIA-CAN cable. The arrow points to pin 1 of the 5-pin screw terminal block. All of the signals on the 5-pin Combicon-style pluggable screw terminal are connected directly to the corresponding pins on the 9-pin D-Sub.

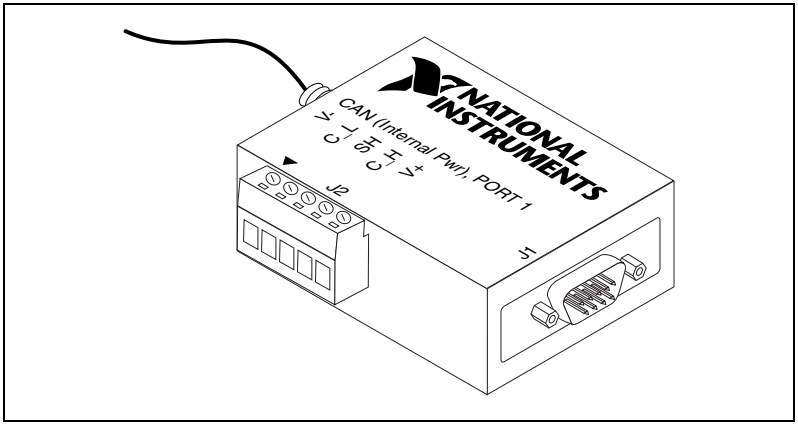
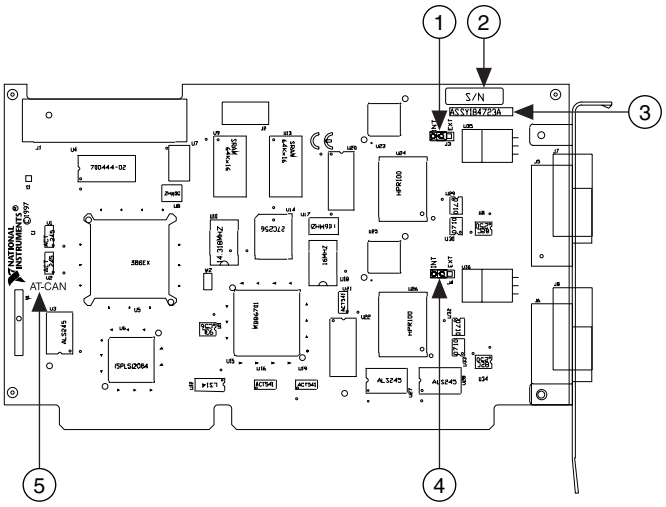


Figure D-3. PCMCIA-CAN Cable

Power Supply Information for the High-Speed CAN Ports

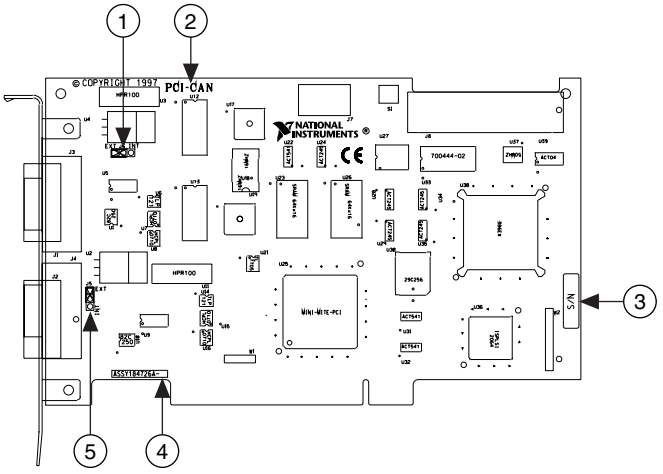
For the AT-CAN, PCI-CAN, and PXI-846*x* series boards, the power source for the CAN physical layer is configured with a jumper. For the AT-CAN and port one of the AT-CAN/2, power is configured with jumper J3. For port two of the AT-CAN/2, power is configured with jumper J4. The location of these jumpers is shown in Figure D-4.



1	Power Supply Jumper J3	3	Assembly Number	5	Product Name
2	Serial Number	4	Power Supply Jumper J4		

Figure D-4. AT-CAN/2 Parts Locator Diagram

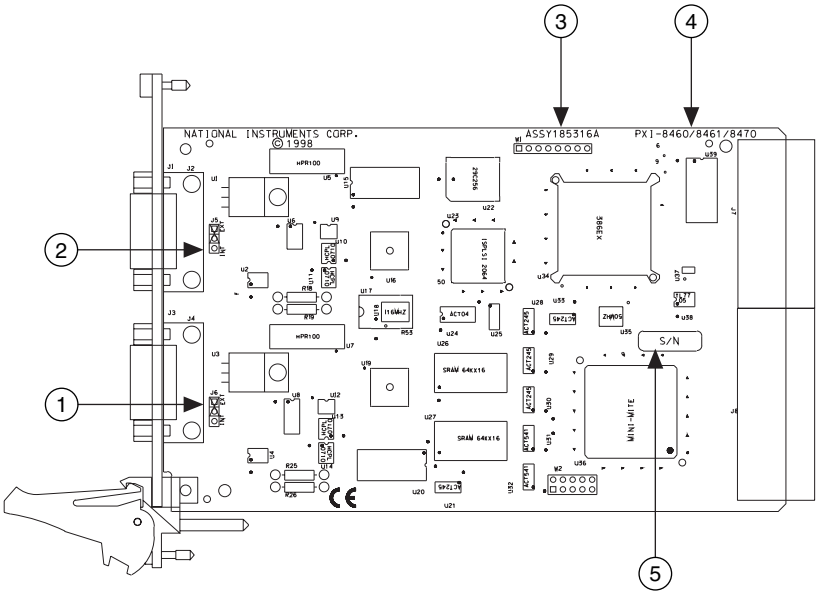
For the PCI-CAN and port one of the PCI-CAN/2 power is configured with jumper J6. For port two of the PCI-CAN/2, power is configured with jumper J5. These jumpers are shown in Figure D-5.



1	Power Supply Jumper J6	3	Serial Number	5	Power Supply Jumper J5
2	Product Name	4	Assembly Number		

Figure D-5. PCI-CAN/2 Parts Locator Diagram

For port one of the PXI-8461, power is configured with jumper J5. For port two of the PXI-8461, power is configured with jumper J6. The location of these jumper is shown in Figure D-6.



1	Power Supply Jumper J6	3	Assembly Number	5	Serial Number
2	Power Supply Jumper J5	4	Product Name		

Figure D-6. PXI-8461 Parts Locator Diagram

Connecting pins 1 and 2 of a jumper configures the CAN physical layer to be powered externally (from the bus cable power). In this configuration, the power must be supplied on the V+ and V– pins on the port connector.

Connecting pins 2 and 3 of a jumper configures the CAN physical layer to be powered internally (from the board). In this configuration, the V– signal serves as the reference ground for the isolated signals.

Figure D-7 shows how to configure your jumpers for internal or external power supplies.

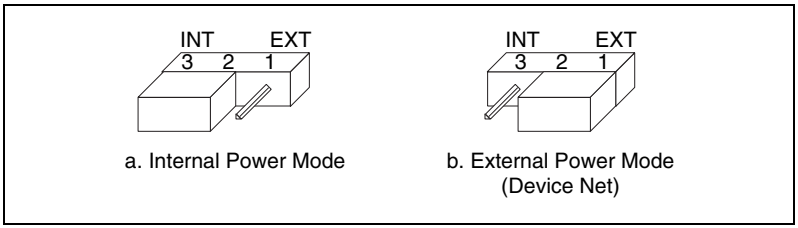


Figure D-7. Power Source Jumpers

The CAN physical layer is still isolated regardless of the power source chosen.

The PCMCIA-CAN series cards are available with two types of cable. The DeviceNet (bus powered) cable requires that the CAN physical layer be powered from the bus cable power.

The internal-powered cable supplies power to the CAN physical layer from the host computer. The V+ pin is not connected to any internal signals, but the corresponding pins on the 9-pin D-Sub and the 5 pin Combicon-style connectors are still connected. The V– pins serves as the reference ground for the isolated signals.

The CAN physical layer is isolated from the computer in both types of cable.

Bus Power Supply Requirements

If the CAN physical layer is powered from a bus power supply, the power supply should be a DC power supply with an output of 10 to 30 V. The power requirements for the CAN ports for Bus-Powered configurations are shown in Table D-1. You should take these requirements into account when determining requirements of the bus power supply for the system.

Table D-1. Power Requirements for the CAN Physical Layer for Bus-Powered Versions

Characteristic	Specification
Voltage requirement	V+ 10-30 VDC
Current requirement	40 mA typical 100 mA maximum

Cable Specifications

Cables should meet the physical medium requirements specified in ISO 11898, shown in Table D-2.

Belden cable (3084A) meets all of those requirements, and should be suitable for most applications.

Table D-2. ISO 11898 Specifications for Characteristics of a CAN_H and CAN_L Pair of Wires

Characteristic	Value
Impedance	108 Ω minimum, 120 Ω nominal, 132 Ω maximum
Length-related resistance	70 mΩ/m nominal
Specific line delay	5 ns/m nominal

Cable Lengths

The allowable cable length is affected by the characteristics of the cabling and the desired bit transmission rates. Detailed cable length recommendations can be found in the ISO 11898, CiA DS 102, and DeviceNet specifications.

ISO 11898 specifies 40 m total cable length with a maximum stub length of 0.3 m for a bit rate of 1 Mb/s. The ISO 11898 specification says that significantly longer cable lengths may be allowed at lower bit rates, but each node should be analyzed for signal integrity problems.

Table D-3 lists the DeviceNet cable length specifications.

Table D-3. DeviceNet Cable Length Specifications

Bit Rate	Thick Cable	Thin Cable
500 kb/s	100 m	100 m
250 kb/s	200 m	100 m
100 kb/s	500 m	100 m

Number of Devices

The maximum number of devices depends on the electrical characteristics of the devices on the network. If all of the devices meet the requirements of ISO 11898, at least 30 devices may be connected to the bus. Higher numbers of devices may be connected if the electrical characteristics of the devices do not degrade signal quality below ISO 11898 signal level specifications. If all of the devices on the network meet the DeviceNet specifications, 64 devices may be connected to the network.

Cable Termination

The pair of signal wires (CAN_H and CAN_L) constitutes a transmission line. If the transmission line is not terminated, each signal change on the line causes reflections that may cause communication failures.

Because communication flows both ways on the CAN bus, CAN requires that both ends of the cable be terminated. However, this requirement does not mean that every device should have a termination resistor. If multiple devices are placed along the cable, only the devices on the ends of the cable should have termination resistors. See Figure D-8 for an example of where termination resistors should be placed in a system with more than two devices.

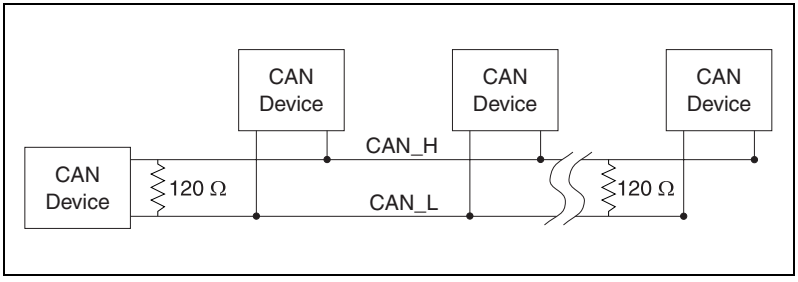


Figure D-8. Termination Resistor Placement

The termination resistors on a cable should match the nominal impedance of the cable. ISO 11898 requires a cable with a nominal impedance of 120 Ω; therefore, a 120 Ω resistor should be used at each end of the cable. Each termination resistor should be capable of dissipating 0.25 W of power.

Cabling Example

Figure D-9 shows an example of a cable to connect two CAN devices. For the internal power configuration, no V+ connection is required.

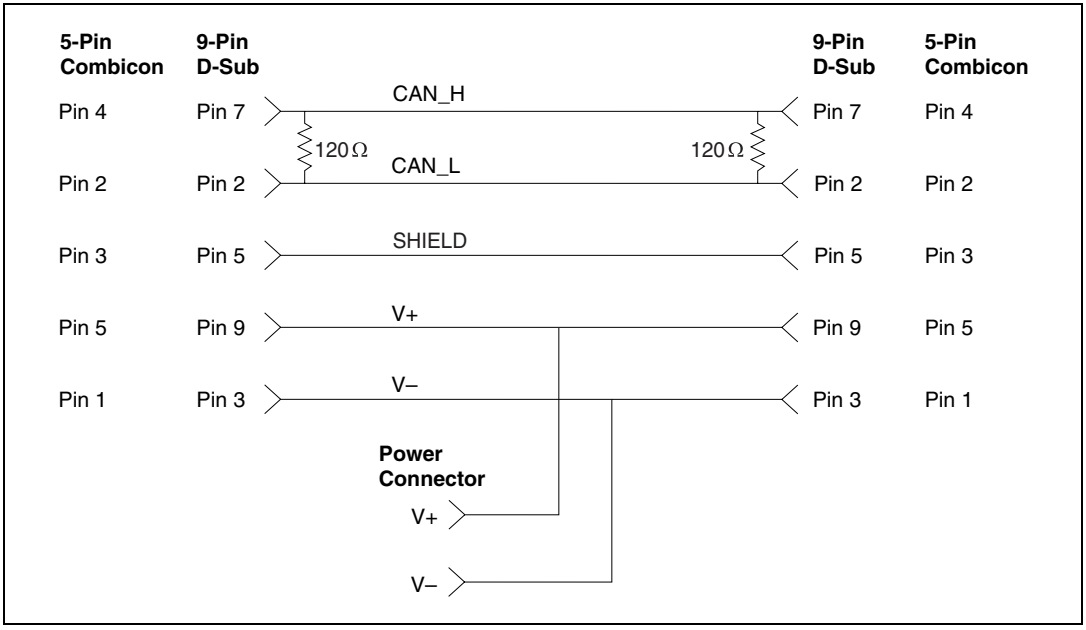


Figure D-9. Cabling Example

E

Cabling Requirements for Low-Speed CAN

This appendix describes the cabling requirements for the low-speed CAN hardware.

Cables should be constructed to meet these requirements, as well as the requirements of the other CAN devices in the network.

Connector Pinouts

The low-speed CAN hardware has DB-9 D-Sub connector(s). The 9-pin D-Sub follows the pinout recommended by CiA DS 102. Figure E-1 shows the pinout for this connector.

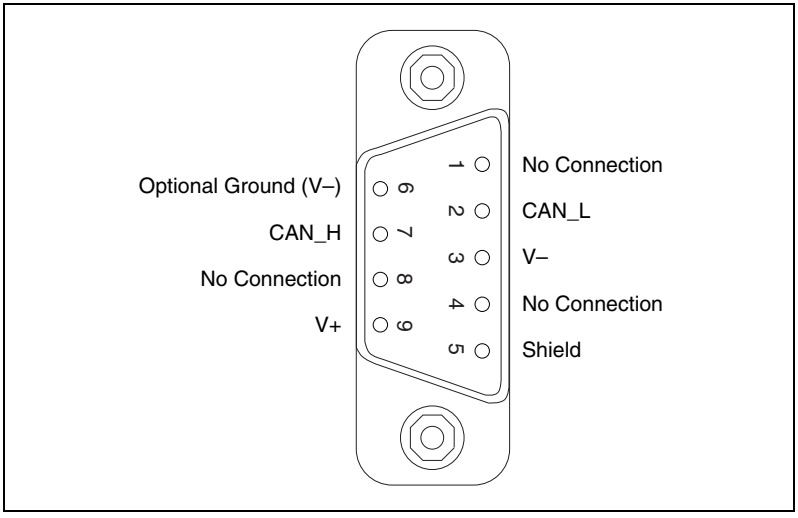


Figure E-1. Pinout for 9-Pin D-Sub Connector

CAN_H and CAN_L are signal lines that carry the data on the CAN network. These signals should be connected using twisted-pair cable.

The V+ and V– pins are used to supply bus power to the CAN physical layer if external power is required for the CAN physical layer. If internal power for the CAN physical layer is used, the V- pin serves as the reference ground for CAN_H and CAN_L. See the next section, *Power Supply Information for the Low-Speed CAN Ports*, for more information.

Figure E-2 shows the end of a PCMCIA-CAN/LS cable. The arrow points to pin 1 of the 7-pin screw terminal block. All of the signals on the 7-pin pluggable screw terminal, except RTL and RTH, are connected directly to the corresponding pins on the 9-pin D-Sub.

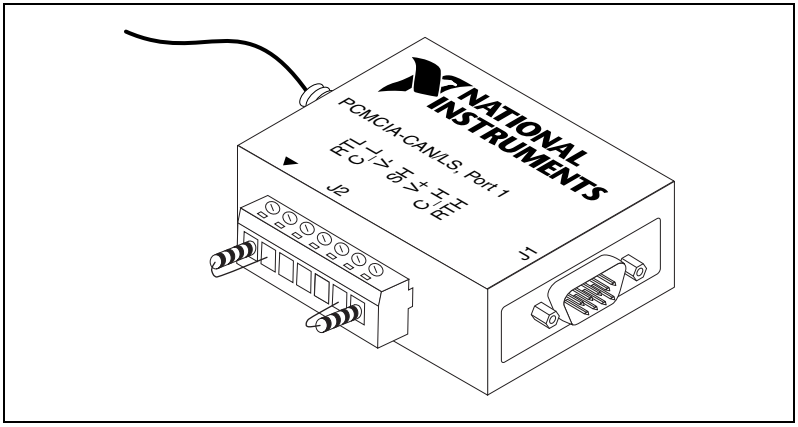
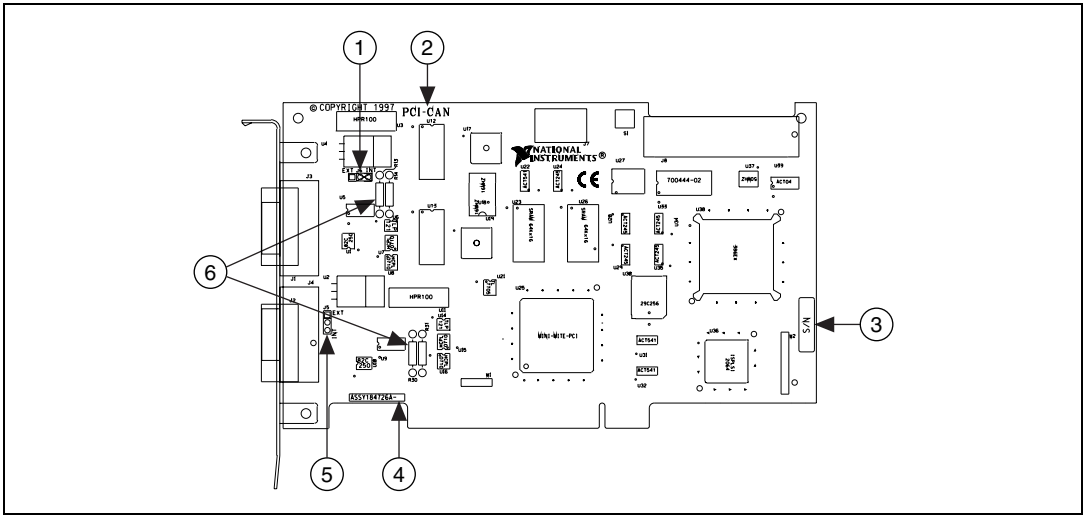


Figure E-2. PCMCIA-CAN/LS Cable

Power Supply Information for the Low-Speed CAN Ports

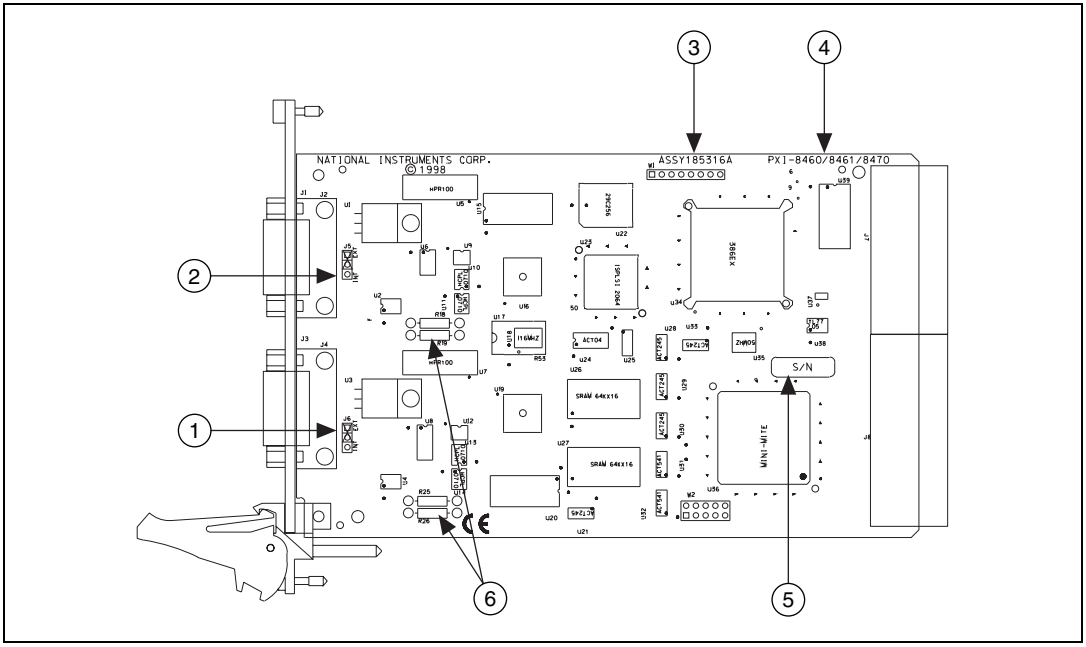
For the PCI-CAN/LS and port one of the PCI-CAN/LS2, power is configured with jumper J6. For port two of the PCI-CAN/LS2, power is configured with jumper J5. These jumpers are shown in Figure E-3.



1	Power Supply Jumper J6	3	Serial Number	5	Power Supply Jumper J5
2	Product Name	4	Assembly Number	6	Termination Resistor Sockets

Figure E-3. PCI-CAN/LS2 Parts Locator Diagram

For port one of the PXI-8460, power is configured with jumper J5. For port two of the PXI-8460, power is configured with jumper J6. These jumpers are shown in Figure E-4.



1	Power Supply Jumper J6	3	Assembly Number	5	Serial Number
2	Power Supply Jumper J5	4	Product Name	6	Termination Resistor Sockets

Figure E-4. PXI-8460 Parts Locator Diagram

Connecting pins 1 and 2 of a jumper configures the CAN physical layer to be powered externally (from the bus cable power). In this configuration, the power must be supplied on the V+ and V– pins on the port connector.

Connecting pins 2 and 3 of a jumper configures the CAN physical layer to be powered internally (from the board). In this configuration, the V– signal serves as the reference ground for the isolated signals. Even if the CAN physical layer is powered internally, the fault-tolerant CAN transceiver still requires bus power to be supplied in order for it to monitor the power supply (battery) voltage.

Figure E-5 shows how to configure your jumpers for internal or external power supplies.

a. Internal Power Mode b. External Power Mode

Figure E-5. Power Source Jumpers

The CAN physical layer is still isolated regardless of the power source chosen.

Bus Power Supply Requirements

If the CAN physical layer is powered from a bus power supply, the power supply should be a DC power supply with an output of 8 to 27 V. The power requirements for the CAN ports for Bus-Powered configurations are shown in Table E-1. You should take these requirements into account when determining requirements of the bus power supply for the system.

Table E-1. Power Requirements for the Low-Speed CAN Physical Layer for Bus-Powered Versions

Characteristic	Specification
Voltage requirement	V+ 8-27 VDC
Current requirement	40 mA typical 100 mA maximum

Cable Specifications

Cables should meet the physical medium requirements specified in ISO 11519-2, shown in Table E-2.

Table E-2. ISO 11519-2 Specifications for Characteristics of a CAN_H and CAN_L Pair of Wires

Characteristic	Value
Length-related resistance	90 mΩ/m nominal
Length-related capacitance: CAN_L and ground, CAN_H and ground, CAN_L and CAN_H	30 pF/m nominal

Belden cable (3084A) meets all of those requirements, and should be suitable for most applications.

Number of Devices

The maximum number of devices depends on the electrical characteristics of the devices on the network. If all of the devices meet the requirements of ISO 11519-2, at least 20 devices may be connected to the bus. Higher numbers of devices may be connected if the electrical characteristics of the devices do not degrade signal quality below ISO 11519-2 signal level specifications.

Low-Speed Termination

Every device on the low-speed CAN network requires a termination resistor for each CAN data line: R_{RTH} for CAN_H and R_{RTL} for CAN_L. Figure E-6 shows termination resistor placement in a low-speed CAN network.

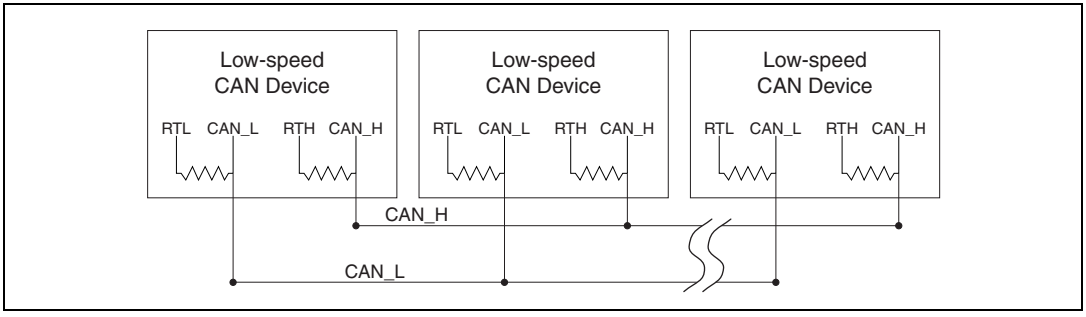


Figure E-6. Termination Resistor Placement for Low-Speed CAN

The following sections explain how to determine the correct resistor values for your low-speed CAN board, and how to replace those resistors, if necessary.

Determining the Necessary Termination Resistance for Your Board

Unlike high-speed CAN, low-speed CAN requires termination at the low-speed CAN transceiver instead of on the cable. The termination requires one resistor: RTH for CAN_H and RTL for CAN_L. This configuration allows the Philips fault-tolerant CAN transceiver to detect any of seven network faults. You can use your PCI-CAN/LS or PXI-8460 to connect to a low-speed CAN network having from two to 32 nodes as specified by Philips (including the port on the PCI-CAN/LS or PXI-8460 as a node). You can also use the PCI-CAN/LS or PXI-8460 to communicate with individual low-speed CAN devices. It is important to determine the overall termination of your existing network, or the termination of your individual device, before connecting it to a PCI-CAN/LS or PXI-8460 port. Philips recommends an overall RTH and RTL termination of 100 to 500 Ω (each) for a properly terminated low-speed network. The overall network termination may be determined as follows:

$$\frac{1}{R_{\text{RTH overall}^{\dagger}}} = \frac{1}{R_{\text{RTH node 1}}} + \frac{1}{R_{\text{RTH node 2}}} + \frac{1}{R_{\text{RTH node 3}}} + \frac{1}{R_{\text{RTH node n}}}$$

Philips also recommends an individual device RTH and RTL termination of 500 to 16 kΩ. The PCI-CAN/LS or PXI-8460 board ships with mounted RTH and RTL values of 510 Ω ±5% per port. The PCI-CAN/LS or PXI-8460 kit also includes a pair of 15 kΩ ±5% resistors for each port. After determining the termination of your existing network or device, you can use the following formula to indicate which value should be placed on your PCI-CAN/LS or PXI-8460 board in order to produce the proper overall RTH and RTL termination of 100 to 500 Ω upon connection of the board:

$$R_{\text{RTH overall}^{*\dagger}} = \frac{1}{\left(\frac{1}{R_{\text{RTH of low-speed CAN interface}^{**}}} + \frac{1}{R_{\text{RTH of existing network or device}}}\right)}$$

*$R_{\text{RTH overall}}$ should be between 100 and 500 Ω

**$R_{\text{RTH of low-speed CAN interface}}$= 510 Ω ±5% (mounted) or 15 kΩ ±5% (in kit)

†$R_{\text{RTH}} = R_{\text{RTL}}$

As the formula indicates, the 510 Ω ±5% shipped on your board will work with properly terminated networks having a total RTH and RTL termination of 125 to 500 Ω, or individual devices having an RTH and RTL termination of 500 to 16 kΩ. For communication with a network having an overall RTH and RTL termination of 100 to 125 Ω, you will need to replace the 510 Ω resistors with the 15 kΩ resistors in the kit. Please refer to the next section, *Replacing the Termination Resistors on Your PCI-CAN/LS Board*.

The PCMCIA-CAN/LS cable ships with screw-terminal mounted RTH and RTL values of 510 Ω ±5% per port. The PCMCIA-CAN/LS cable also internally mounts a pair of 15.8 KΩ ±1% resistors in parallel with the external 510 Ω resistors for each port. This produces an effective RTH and RTL of 494 Ω per port for the PCMCIA-CAN/LS cable. After determining the termination of your existing network or device, you can use the formula below to indicate which configuration should be used on your PCMCIA-CAN/LS cable to produce the proper overall RTH and RTL termination of 100 to 500 Ω upon connection of the cable:

$$R_{\text{RTH overall}*\dagger} = \frac{1}{\left(\frac{1}{R_{\text{RTH of PCMCIA-CAN/LS}**}} + \frac{1}{R_{\text{RTH of existing network or device}}}\right)}$$

$*R_{\text{RTH overall}}$ should be between 100 and 500 Ω

$**R_{\text{RTH of PCMCIA-CAN/LS}}$ = 494 Ω (510 Ω ±5% (external) in parallel with 15.8KΩ ±1% (internal)), or 15.8KΩ ±1% (internal) only

$\dagger R_{\text{RTH}} = R_{\text{RTL}}$

As the formula indicates, the 510 Ω ±5% in parallel with 15.8 KΩ ±1% shipped on your cable will work with properly terminated networks having a total RTH and RTL termination of 125 to 500 Ω, or individual devices having an RTH and RTL termination of 500 to 16 KΩ. For communication with a network having an overall RTH and RTL termination of 100 to 125 Ω, you will need to disconnect the 510 Ω resistors from the 7-pin pluggable screw terminal. This will make the RTH and RTL values of the PCMCIA-CAN/LS cable equal to the internal resistance of 15.8 KΩ ±1%. To produce RTH and RTL values between 494 and 15.8 KΩ on the PCMCIA-CAN/LS cable, use the following formula:

$$R_{\text{External RTH of PCMCIA-CAN/LS}^{\dagger}} = \frac{1}{\left(\frac{1}{R_{\text{Desired RTH of PCMCIA-CAN/LS}}} - \frac{1}{R_{\text{Internal RTH of PCMCIA-CAN/LS}***}}\right)}$$

$^{***}R_{\text{Internal RTH of PCMCIA-CAN/LS}} = 15.8\ \text{K}\Omega \pm 1\%$

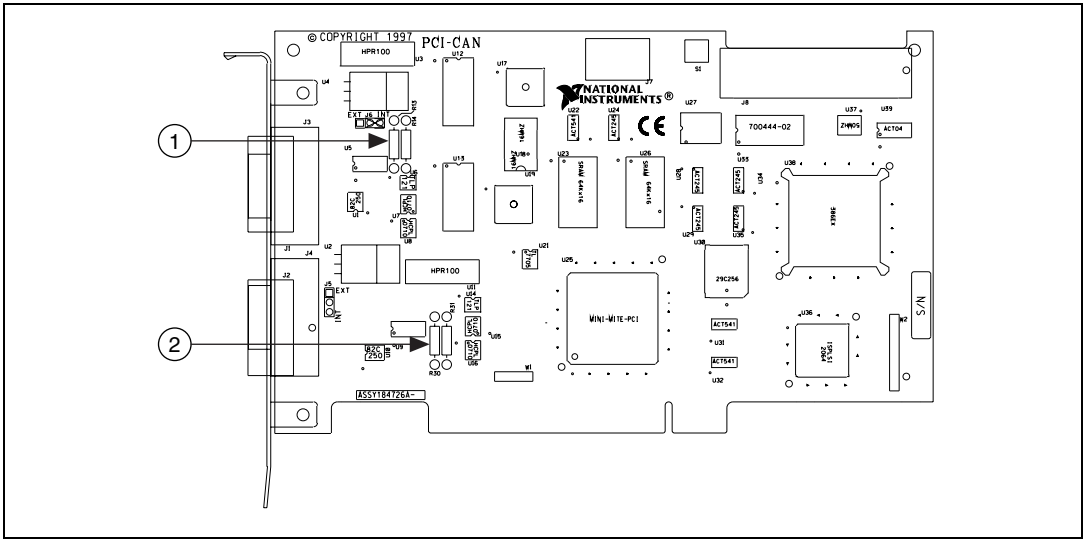
$^{\dagger}R_{\text{RTH}} = R_{\text{RTL}}$

For information on replacing the external RTH and RTL resistors on your PCMCIA-CAN/LS cable, refer to *Replacing the Termination Resistors on Your PCMCIA-CAN/LS Cable*.

Replacing the Termination Resistors on Your PCI-CAN/LS Board

Follow these steps to replace the termination resistors on your PCI-CAN/LS board, after you have determined the correct value in the previous section, *Determining the Necessary Termination Resistance for Your Board*.

1. Remove the termination resistors on your low-speed CAN board. Figure E-7 shows the location of the termination resistor sockets on a PCI-CAN/LS2 board.



1	Port 1 Termination Resistors	2	Port 2 Termination Resistors

Figure E-7. Location of Termination Resistors on PCI-CAN/LS2 Board

2. Cut and bend the lead wires of the resistors you want to install. Refer to Figure E-8.

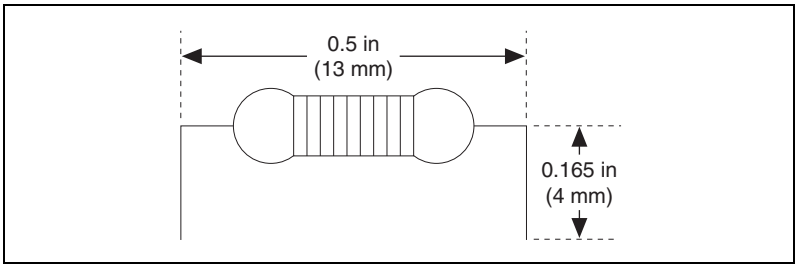


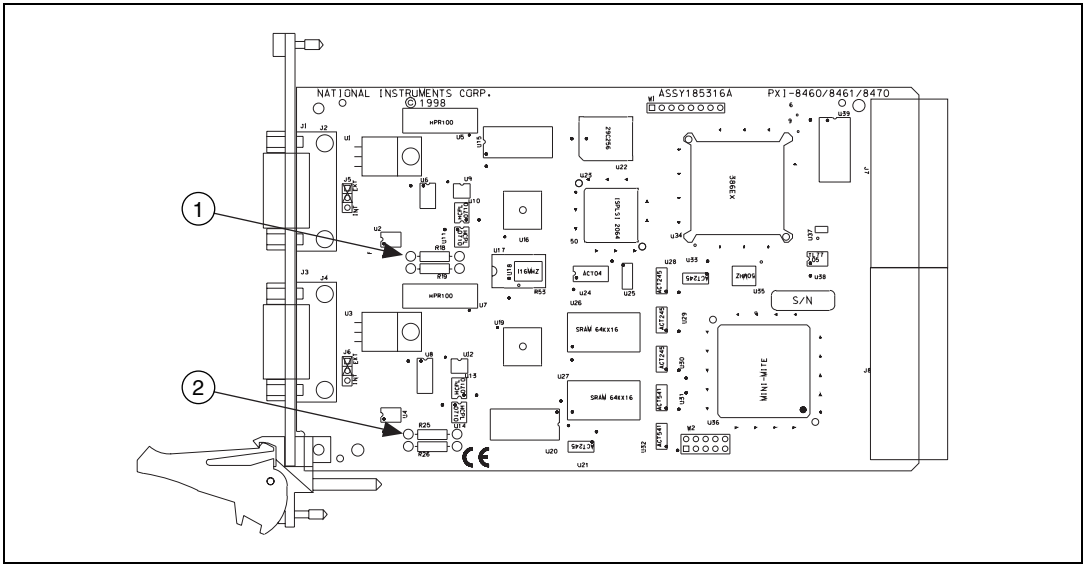
Figure E-8. Preparing Lead Wires of Replacement Resistors

3. Insert the replacement resistors into the empty sockets.
4. Refer to the *Installation Guide, CAN Hardware and the NI-CAN Software for Windows 2000/NT/Me/9*x in the jewel case of your program CD to complete the hardware installation.

Replacing the Termination Resistors on Your PXI-8460 Board

Follow these steps to replace the termination resistors, after you have determined the correct value in the previous section, *Determining the Necessary Termination Resistance for Your Board*.

1. Remove the termination resistors on your PXI-8460. Figure E-9 shows the location of the termination resistor sockets on a PXI-8460.



1 Port 1 Termination Resistors	2 Port 2 Termination Resistors

Figure E-9. Location of Termination Resistors on a PXI-8460

2. Cut and bend the lead wires of the resistors you want to install. Refer to Figure E-10.

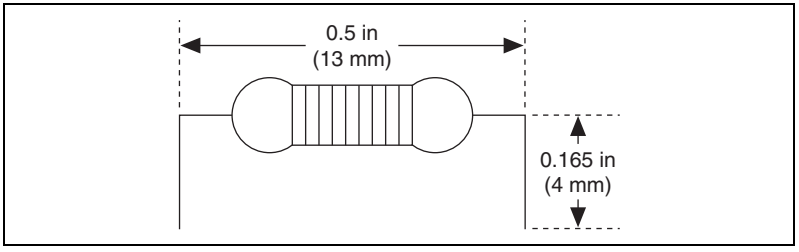


Figure E-10. Preparing Lead Wires of Replacement Resistors

3. Insert the replacement resistors into the empty sockets.
4. Refer to the *Installation Guide, CAN Hardware and the NI-CAN Software for Windows 2000/NT/Me/9*x in the jewel case of your program CD to complete the hardware installation.

Replacing the Termination Resistors on Your PCMCIA-CAN/LS Cable

Follow these steps to replace the termination resistors on your PCMCIA-CAN/LS cable after you have determined the correct value in the section *Determining the Necessary Termination Resistance for Your Board*.

1. Remove the two termination resistors on your PCMCIA-CAN/LS cable by loosening the pluggable terminal block mounting screws for pins 1 and 2 (RTL) and pins 6 and 7 (RTH).
2. Bend and cut the lead wires of the two resistors you want to install, as shown Figure E-11.

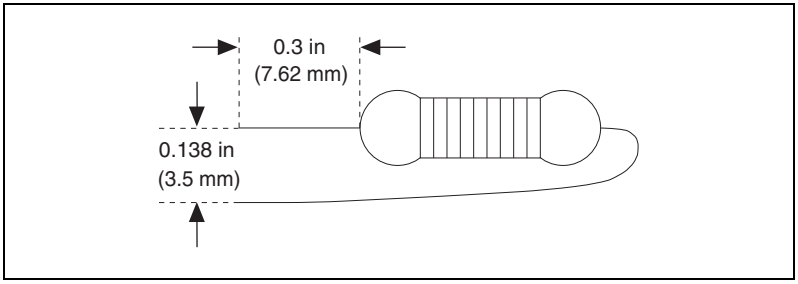


Figure E-11. Preparing Lead Wires of PCMCIA-CAN/LS Cable Replacement Resistors

3. Mount RTL by inserting the leads of one resistor into pins 1 and 2 of the pluggable terminal block and tightening the mounting screws. Mount RTH by inserting the leads of the second resistor into pins 6 and 7 of the pluggable terminal block and tightening the mounting screws.
4. Refer to the *Installation Guide, CAN Hardware and the NI-CAN Software for Windows 2000/NT/Me/9*x in the jewel case of your program CD to complete the hardware installation.

Cabling Example

Figure E-12 shows an example of a cable to connect two low-speed CAN devices. For the PCMCIA-CAN/LS cables, only V-, CAN_L, and CAN _H are required to be connected to the bus.

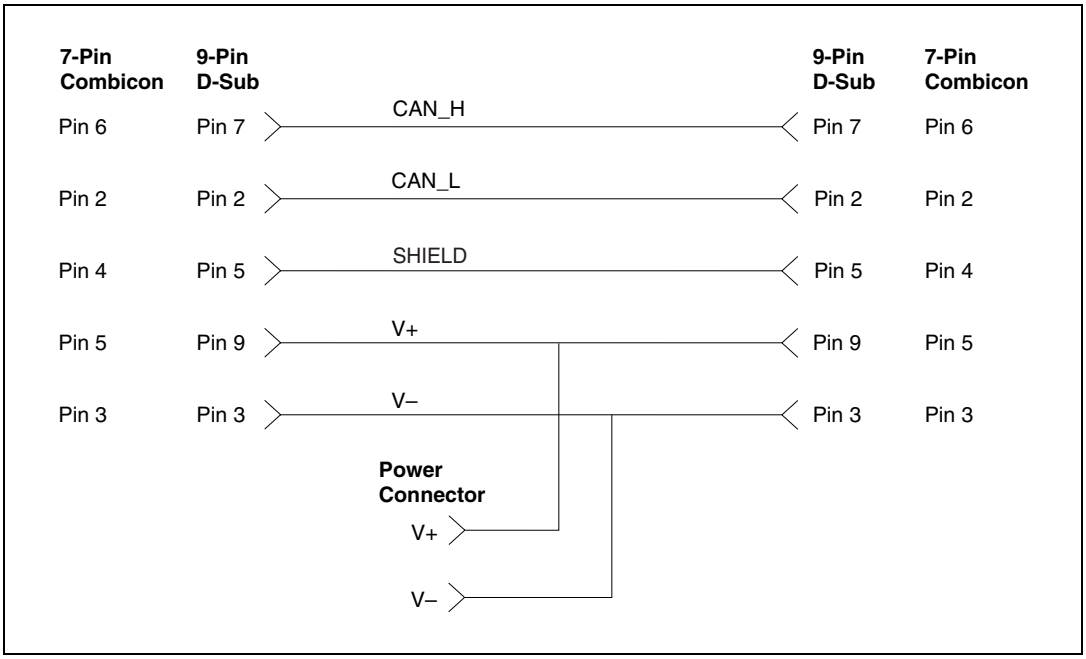


Figure E-12. Cabling Example

F

Cabling Requirements for Dual-Speed CAN

This section describes the cabling requirements for the dual-speed CAN hardware.

Port Identification

The PCI-CAN/DS board, PXI-8462 board, and PCMCIA-CAN/DS cable each provide a high-speed CAN port (port one), and a low-speed CAN port (port two). Port one of the PCI-CAN/DS is identical to port one of the PCI-CAN and PCI-CAN/2, and port two is identical to port two of the PCI-CAN/LS2.

Port one of the PXI-8462 is identical to port one of the PXI-8461 one-port and PXI-8461 two-port boards. Port two of the PXI-8462 is identical to port two of the PXI-8460 two-port board.

Port one of the PCMCIA-CAN/DS cable is identical to port one of the PCMCIA-CAN and PCMCIA-CAN/2 cables, and port two is identical to port two of the PCMCIA-CAN/LS2 cable. The PCI-CAN/DS board, PXI-8462 board and PCMCIA-CAN/DS cable allow simultaneous communication with both a high-speed and low-speed bus, each with its own specific cabling and termination requirements. For cabling requirements and port information for the high-speed CAN port, please refer to Appendix D, *Cabling Requirements for High-Speed CAN*, in this manual. For cabling requirements and port information for the low-speed CAN port, please refer to Appendix E, *Cabling Requirements for Low-Speed CAN*, in this manual.

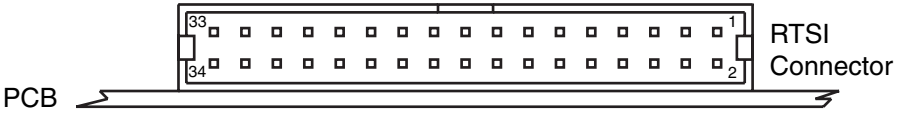
G

RTSI Bus

This appendix describes the RTSI interface on your CAN board.

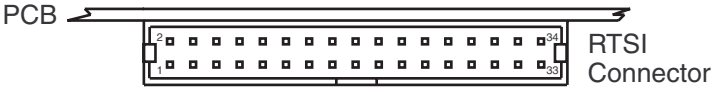
RTSI, AT and PCI

Figure G-1 shows the RTSI connector pinout for the AT-CAN series boards. Figure G-2 shows the RTSI connector pinout for the PCI-CAN series boards.



AT-CAN Series Trigger	AT-CAN Series Pin Number
RTSI Trigger <0>	15
RTSI Trigger <1>	13
RTSI Trigger <2>	11
RTSI Trigger <3>	9
RTSI Trigger <4>	7
RTSI Trigger <5>	5
RTSI Trigger <6>	3
RTSI Oscillator	1
GND	2, 4, 6, 8, 10, 12, 14

Figure G-1. AT-CAN Series RTSI Connector Pinout



PCI-CAN Series Trigger	PCI-CAN Series Pin Number
RTSI Trigger <0>	20
RTSI Trigger <1>	22
RTSI Trigger <2>	24
RTSI Trigger <3>	26
RTSI Trigger <4>	28
RTSI Trigger <5>	30
RTSI Trigger <6>	32
RTSI Oscillator	34
GND	19, 21, 23, 25, 27, 29, 31, 33

Figure G-2. PCI-CAN Series RTSI Connector Pinout

Using the National Instruments RTSI bus with your CAN board consists of connecting it to other RTSI-equipped boards with RTSI ribbon cable, to route timing and trigger signals between the boards. Using the RTSI bus, your CAN board can be synchronized with multiple National Instruments DAQ boards in your computer. The RTSI bus can also be used to synchronize multiple CAN boards.

The AT-CAN, AT-CAN/2, PCI-CAN and PCI-CAN/2 boards allow for the connection of four RTSI input signals and four RTSI out put signals. In order to fully support the fault reporting capabilities of the low-speed transceivers used on the PCI-CAN/LS, PCI-CAN/LS2, and PCI-CAN/DS, three RTSI lines on those boards are reserved for low-speed CAN fault reporting. This allows for the connection of three RTSI input signals and two RTSI output signals to the boards, providing them the real time synchronization benefits of RTSI without sacrificing low-speed CAN fault reporting.

RTSI, PXI and CompactPCI

Using PXI-compatible products with standard CompactPCI products is an important feature provided by the *PXI Specification*, Revision 1.0. If you use a PXI-compatible plug-in device in a standard CompactPCI chassis, you will be unable to use PXI-specific functions, but you can still use the basic plug-in device functions. For example, the RTSI bus on your PXI-846*x* series board is available in a PXI chassis, but not in a CompactPCI chassis. The CompactPCI specification permits vendors to develop sub-buses that coexist with the basic PCI interface on the CompactPCI bus. Compatible operation is not guaranteed between CompactPCI devices with different sub-buses nor between CompactPCI devices with sub-buses and PXI. The standard implementation for CompactPCI does not include these sub-buses. Your PXI-846*x* device will work in any standard CompactPCI chassis adhering to the *PICMG 2.0 R2.1 CompactPCI* core specification using the 64-bit definition for J2. PXI specific features are implemented on the J2 connector of the CompactPCI bus. Table G-1 lists the J2 pins your PXI-846*x* series board uses. Your PXI board is compatible with any CompactPCI chassis with a sub-bus that does not drive these lines. Even if the sub-bus is capable of driving these lines, the board is still compatible as long as those pins on the sub-bus are disabled by default and not ever enabled. Damage may result if these lines are driven by the sub-bus.

The PXI-8461 one-port and two-port boards allow for the connection of four RTSI input signals and four RTSI output signals. In order to fully support the fault reporting capabilities of the low-speed transceivers used on the PXI-8460 one port, PXI-8460 two port, and PXI-8462, three RTSI lines on those boards are reserved for low-speed CAN fault reporting. This allows for the connection of three RTSI input signals and two RTSI output signals to the boards, providing them the real time synchronization benefits of RTSI without sacrificing low-speed CAN fault reporting.

Table G-1. Pins Used By the PXI-846*x* Series Boards

PXI Pin Name	PXI J2 Pin Number
PXI Star	D17
PXI Trigger <0>	B16
PXI Trigger <1>	A16
PXI Trigger <2>	A17

Table G-1. Pins Used By the PXI-846*x* Series Boards (Continued)

PXI Pin Name	PXI J2 Pin Number
PXI Trigger <3>	A18
PXI Trigger <4>	B18
PXI Trigger <5>	C18
PXI Trigger <7>	E16

RTSI Cables

National Instruments offers a variety of RTSI bus cables for connecting your CAN board to other CAN or DAQ hardware. For more specific information about these cables, you can refer to the National Instruments catalog, or our Web site `ni.com`, or call the National Instruments office nearest you listed in the *Worldwide Support* section of Appendix I, *Technical Support Resources*.

RTSI Programming

For more information on RTSI programming, refer to the *NI-CAN Programmer Reference Manual.* Refer to *RTSI Bus Overview* in Chapter 1 of this manual for more information on the RTSI bus.

H

Specifications

This appendix describes the physical characteristics of the CAN hardware, along with the recommended operating conditions.

AT-CAN Series

Dimensions..... 10.67 by 16.51 cm (4.2 by 6.5 in.)

Power requirement ..... +5 VDC, 500 mA typical

I/O Connector..... 9-pin D-Sub for each port (standard)
or
5-pin Combicon-style pluggable DeviceNet screw terminal

Operating environment

- Component temperature..... 0 to 55 °C
- Relative humidity..... 10% to 90%, noncondensing

Storage environment

- Temperature ..... –20 to 70 °C
- Relative humidity..... 5% to 90%, noncondensing

PCI-CAN Series

Dimensions..... 10.67 by 17.46 cm (4.2 by 6.88 in.)

Power requirement ..... +5 VDC, 775 mA typical

I/O connector..... 9-pin D-Sub for each port (standard)
or
5-pin Combicon-style pluggable DeviceNet screw terminal (high-speed CAN only)

Operating environment

Ambient temperature .....0 to 55 °C

Relative humidity .....10% to 90%, noncondensing

Storage environment

Temperature.....–20 to 70 °C

Relative humidity .....5% to 90%, noncondensing

PCMCIA-CAN Series

Dimensions .....8.56 by 5.40 by 0.5 cm (3.4 by 2.1 by 0.4 in.)

Power requirement.....500 mA typical

I/O connector .....Cable with 9-pin D-Sub and pluggable screw terminal for each port

Operating environment

Component temperature .....0 to 55 °C

Relative humidity .....10% to 90%, noncondensing

Storage Environment

Temperature.....–20 to 70 °C

Relative humidity .....5% to 90%, noncondensing

High-Speed CAN Port Characteristics

Bus Power.....0 to 30 V, 40 mA typical 100mA maximum

CAN-H, CAN-L .....–8 to +18V, DC or peak, CATI

Low-Speed CAN Port Characteristics

Bus Power.....8 to 27 V, 40 mA typical 100 mA maximum

CAN-H, CAN-L .....–10 to +27V, DC or peak, CATI

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
k-	kilo-	10^{3}

Symbols

°	degrees
Ω	ohms
%	percent

A

A	amperes
AC	alternating current
action	*See* method.
actuator	A device that uses electrical, mechanical, or other signals to change the value of an external, real-world variable. In the context of device networks, actuators are devices that receive their primary data value from over the network; examples include valves and motor starters. Also known as *final control element.*
ANSI	American National Standards Institute
Application Programming Interface (API)	A collection of functions used by a user application to access hardware. Within NI-CAN, you use API functions to make calls into the NI-CAN driver.
arbitration ID	An 11- or 29-bit ID transmitted as the first field of a CAN frame. The arbitration ID determines the priority of the frame, and is normally used to identify the data transmitted in the frame.
AT-compatible	compatible with the 16-bit Industry Standard Architecture

attribute	The externally visible qualities of an object; for example, an instance Mike of class Human could have the attributes Sex and Age, with the values Male and 31. Also known as *property*.

B

b	bits
B	bytes
bus off	A CAN node goes into the bus off state when its transmit error counter increments above 255. The node does not participate in network traffic, because it assumes that a defect exists that must be corrected.

C

C	Celsius
CAN	Controller Area Network
CAN data frame	Frame used to transmit the actual data of a CAN Object. The RTR bit is clear, and the data length indicates the number of data bytes in the frame.
CAN/DS	Dual-speed CAN
CAN frame	In addition to fields used for error detection/correction, a CAN frame consists of an arbitration ID, an Identifier Extension, SOF and EOF bits, the RTR bit, a four-bit Data Length Code, and zero to eight bytes of data.
CAN/LS	*See* Low-speed CAN.
CAN Network Interface Object	Within NI-CAN, an object that encapsulates a CAN network interface on the host computer.
CAN Object	A CAN identifier, along with its associated data.
CAN remote frame	Frame used to request data for a CAN Object from a remote node; the RTR bit is set, and the data length indicates the amount of data desired (but no data bytes are included).
CiA	CAN in Automation
class	A set of objects that share a common structure and a common behavior.

connection	An association between two or more nodes on a network that describes when and how data is transferred.
controller	A device that receives data from sensors and sends data to actuators in order to hold one or more external, real-world variables at a certain level or condition. A thermostat is a simple example of a controller.

D

DC	direct current
device	*See* node.
device network	Multi-drop digital communication network for sensors, actuators, and controllers.
DLL	dynamic link library
DMA	direct memory access

E

EMI	electromagnetic interference
error active	A CAN node is in error active state when both the receive and transmit error counters are below 128.
error counters	Every CAN node keeps a count of how many receive and transmit errors have occurred. The rules for how these counters are incremented and decremented are defined by the CAN protocol specification.
error passive	A CAN node is in error passive state when one or both of its error counters increment above 127. This state is a warning that a communication problem exists, but the node is still participating in network traffic.
extended arbitration ID	A 29-bit arbitration ID. Frames that use extended IDs are often referred to as CAN 2.0 Part B (the specification that defines them).

F

FCC	Federal Communications Commission
frame	A unit of information transferred across a network from one node to another; the protocol defines the meaning of the bit fields within a frame. Also known as *packet*.
ft	feet

H

hex	hexadecimal.
HMI	Human Machine Interface
Hz	Hertz

I

IEEE	Institute of Electrical and Electronic Engineers
in.	inches
instance	An abstraction of a specific real-world thing; for example, Mike is an instance of the class Human. Also known as *object*.
IRQ	interrupt request
ISA	Industry Standard Architecture
ISO	International Standards Organization

K

KB	Kilobytes of memory

L

LED	Light-emitting Diode
local	Within NI-CAN, anything that exists on the same host (personal computer) as the NI-CAN driver.
Low-speed CAN	Implementation of CAN as defined in ISO 11519.

M

m	meters
MB	Megabytes of memory
method	An action performed on an instance to affect its behavior; the externally visible code of an object. Within NI-CAN, you use NI-CAN functions to execute methods for objects. Also known as *service*, *operation*, and *action*.
minimum interval	For a given connection, the minimum amount of time between subsequent attempts to transmit frames on the connection. Some protocols use minimum intervals to guarantee a certain level of overall network performance.
multi-drop	A physical connection in which multiple devices communicate with one another along a single cable.

N

network interface	A node's physical connection onto a network.
NI-CAN driver	Device driver and/or firmware that implement all the specifics of a CAN network interface. Within NI-CAN, this software implements the CAN Network Interface Object as well as all objects above it in the object hierarchy.
node	A physical assembly, linked to a communication line (cable), capable of communicating across the network according to a protocol specification. Also known as *device*.

notification	Within NI-CAN, an operating system mechanism that the NI-CAN driver uses to communicate events to your application. You can think of a notification of as an API function, but in the opposite direction.

O

object	*See* instance.
object-oriented	A software design methodology in which classes, instances, attributes, and methods are used to hide all of the details of a software entity that do not contribute to its essential characteristics.

P

PC	Personal Computer
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
peer-to-peer	Network connection in which data is transmitted from the source to its destination(s) without need for an explicit request. Although data transfer is generally unidirectional, the protocol often uses low level acknowledgments and error detection to ensure successful delivery.
periodic	Connections that transfer data on the network at a specific rate.
PLC	Programmable Logic Controller
polled	Request/response connection in which a request for data is sent to a device, and the device sends back a response with the desired value.
protocol	A formal set of conventions or rules for the exchange of information among nodes of a given network.
PXI	PCI eXtensions for Instrumentation

R

RAM	Random-access Memory
remote	Within NI-CAN, anything that exists in another node of the device network (not on the same host as the NI-CAN driver).

Remote Transmission Request (RTR) bit	This bit follows the arbitration ID in a frame and indicates whether the frame is the actual data of the CAN Object (CAN data frame) or whether the frame is a request for the data (CAN remote frame).
request/response	Network connection in which a request is transmitted to one or more destination nodes, and those nodes send a response back to the requesting node. In industrial applications, the responding (slave) device is usually a sensor or actuator, and the requesting (master) device is usually a controller. Also known as *master/slave*.
resource	Hardware settings used by National Instruments CAN hardware, including an interrupt request level (IRQ) and an 8 KB physical memory range (such as D0000 to D1FFF hex).
RTSI bus	Real-Time System Integration bus. The National Instruments timing bus that connects CAN and DAQ boards directly (via connectors on top of the PCI-CAN and AT-CAN series boards, and the PXI trigger bus on the PXI-846*x* series boards) for precise synchronization of functions.

S

s	seconds
sensor	A device that measures electrical, mechanical, or other signals from an external, real-world variable; in the context of device networks, sensors are devices that send their primary data value onto the network; examples include temperature sensors and presence sensors. Also known as *transmitter.*
standard arbitration ID	An 11-bit arbitration ID. Frames that use standard IDs are often referred to as CAN 2.0 Part A; standard IDs are by far the most commonly used.

T

trigger	Any event that causes or starts some form of data capture.

U

unsolicited	Connections that transmit data on the network sporadically based on an external event. Also known as *nonperiodic*, *sporadic*, and *event driven*.

V

V	volts
VDC	volts direct current

W

W	Watts
watchdog timeout	A timeout associated with a connection that expects to receive network data at a specific rate. If data is not received before the watchdog timeout expires, the connection is normally stopped. You can use watchdog timeouts to verify that the remote node is still operational.

Index

Numbers

5-pin Combicon-style pluggable screw terminal (figure), D-2
9-pin D-sub connector pinout, high-speed (figure), D-1
9-pin D-sub connector pinout, low-speed (figure), E-1

A

Acknowledgment Bit (ACK) field, 1-5
acknowledgment error, 1-6
application development. *See* programming.
application examples, 4-1
    C/C++ languages, 4-1
    LabVIEW, 4-1
    other programming languages, 4-1
arbitration
    example of CAN arbitration (figure), 1-3
    nondestructive bitwise, 1-2
arbitration ID
    definition, 1-2
    using CAN Objects, 2-5
Arbitration ID field, 1-4
AT-CAN series board, 1-9
    and RTS, 1-10
    AT-CAN/2 parts locator diagram (figure), D-3
    power supply information for high-speed CAN ports, D-3
    specifications, H-1
attributes
    definition, 1-11

B

bit error, 1-6
bus off state, 1-8
bus power supply requirements
    high-speed, D-6
    low-speed, E-5

C

C/C++ languages
    accessing NI-CAN software, 2-2
    application examples, 4-1
    status checking, 2-12
cable lengths, D-7
cable lengths, DeviceNet cable-length specifications (table), D-7
cable specifications, D-7, E-5
    ISO 11519-2 specifications for characteristics of a CAN_H and CAN_L pair of wires (table), E-6
    ISO 11898 specifications for characteristics of a CAN_H and CAN_L pair of wires (table), D-7
cable termination
    high-speed CAN, D-8
    low-speed CAN, E-6
cabling example
    high-speed CAN (figure), D-9
    low-speed CAN (figure), E-13
cabling requirements
    dual-speed CAN, F-1
    high-speed CAN, D-1
    low-speed CAN, E-1
CAN
    See also NI-CAN.
    arbitration, 1-2
    error confinement, 1-6

error detection, 1-5
history and use, 1-1
low speed, 1-8
CAN frame reception flowchart (figure), 3-3
CAN frames
definition, 1-3
fields
Acknowledgment Bit (ACK), 1-5
Arbitration ID, 1-4
Cyclic Redundancy Check (CRC), 1-4
Data Bytes, 1-4
Data Length Code (DLC), 1-4
End of Frame, 1-5
Identifier Extension (IDE), 1-4
Remote Transmit Request (RTR), 1-4
Start of Frame (SOF), 1-3
reading and writing, 2-5
standard and extended formats (figure), 1-3
CAN hardware
determining type installed
Windows 2000, C-4
Windows Me/98/95, A-5
Windows NT, B-3
overview, 1-9
problem encountered in Windows Me/98/95, A-5
CAN identifiers, 1-2
CAN interface cables
5-pin Combicon-style pluggable screw terminal (figure), D-2
cable lengths, D-7
cable termination
high-speed CAN, D-8
low-speed CAN, E-6
cabling example
high-speed CAN (figure), D-9
low-speed CAN (figure), E-13
connector pinouts
high-speed, D-1
low-speed, E-1
RTSI, G-1
connector pinouts, RTSI, G-2
DeviceNet cable-length specifications (table), D-7
dual-speed, F-1
high-speed, D-1
low-speed, E-1
PCMCIA-CAN cable (figure), D-2
PCMCIA-CAN/LS cable
(figure), E-2
termination resistors, E-12
pinout for 9-pin D-sub connector, high-speed (figure), D-1
specifications, D-7
ISO 11898 specifications for characteristics of a CAN_H and CAN_L pair of wires (table), D-7
termination resistor placement
(figure), D-8
low-speed CAN, E-6
CAN interfaces
See also missing CAN interfaces.
number of configurable interfaces
Windows 2000, C-4
Windows Me/98/95, A-5
Windows NT, B-3
CAN Network Interface Objects
communication
starting, 2-9
using objects, 2-9
possible uses, 2-4
using with CAN Objects
flowchart for CAN frame reception (figure), 3-3

CAN Objects
choosing NI-CAN Objects
CAN Network interface Objects, 2-4
CAN Objects, 2-5
closing, 2-10
configuration, methods for, 2-9
definition, 1-2
NI-CAN object hierarchy, 1-12
opening, 2-9
using, 2-5
CAN software. *See* NI-CAN software.
CANopen protocol, 1-11
checking status of function calls. *See* status of function calls, checking.
class, definition, 1-11
closing CAN Objects, 2-10
common questions. *See* troubleshooting and common questions.
communicating with CAN network
starting, 2-9
using objects, 2-9
CompactPCI, PXI, and RTSI, G-3
configuring objects
See also NI-CAN Configuration utility.
calling ncConfig function, 2-9
connector pinouts
high-speed, D-1
low-speed, E-1
RTSI, G-1
connector pinouts, RTSI, G-1, G-2
Controller Area Network. *See* CAN; NI-CAN.
conventions used in manual, *xiii*
CRC error, 1-6
customer education, I-1
Cyclic Redundancy Check (CRC) field, 1-4

D

Data Bytes field, 1-4
data length code (DLC) field, 1-4
detecting state changes, 3-4
Device Manager problems
Windows 2000, C-1
Windows Me/98/95, A-1
device network independence, of NI-CAN software, 1-11
DeviceNet cable-length specifications (table), D-7
DeviceNet protocol, 1-11
direct entry access to NI-CAN software, 2-3
DLC (Data Length Code) field, 1-4
documentation
conventions used in manual, *xiii*
how to use manual set, *xiii*
related documentation, *xiv*
drivers, NI-CAN, 1-14
dual-speed CAN
cabling requirements, F-1

E

embedded processor, 1-10
End of Frame field, 1-5
error cluster (table), 2-11
error confinement
bus off state, 1-8
error active state, 1-7
error passive state, 1-7
error detection
acknowledgement error, 1-6
bit error, 1-6
CRC error, 1-6
form error, 1-6
stuff error, 1-6

F

firmware image files, 1-15
form error, 1-6
frames. *See* CAN frames.
function calls, checking. *See* status of function calls, checking.

functions
GetProcAddress, 2-3
ncAction, 2-9
ncConfig, 2-9
ncCreateNotification, 2-10
ncGetAttribute, 3-4
ncOpenObject, 2-9
ncRead, 2-10
ncWaitForState, 2-10

G

GetProcAddress function, 2-3

H

hardware
overview, 1-9
high-speed CAN
cabling requirements, D-1
port characteristics, H-2
how to use manual set, *xiii*

I

Identifier Extension (IDE) field, 1-4
instance, definition, 1-11
interrupt requirements
Windows 2000, C-4
Windows Me/98/95, A-5
Windows NT, B-3
ISO 11898 standard, 1-1

L

LabVIEW
application examples, 4-1
function library, 2-1
status checking in, 2-11
LabVIEW RT
as a programming method, 2-1
using NI-CAN configuration and diagnostic utilities, 5-3
language interface files, 1-15, 2-2
low-speed CAN
cabling requirements, E-1
port characteristics for bus-powered ports, H-2
preparing lead wires of, (figure), E-11
replacing termination resistors, E-10
termination resistors, E-6
termination resistors, location of, (figure), E-11

M

manual. *See* documentation.
memory resource conflict
Windows Me/98/95, A-3
Windows NT, B-2
methods, definition, 1-11
missing CAN interfaces
Windows 2000
no National Instruments CAN Interface, C-1
not listed in NI-CAN Diagnostic utility, C-3
physically absent interface, C-2
Windows Me/98/95
no National Instruments CAN Interface, A-1
not listed in NI-CAN Diagnostic utility, A-4
physically absent interface, A-2
Windows NT
NI-CAN configuration and diagnostic utilities, B-1
not listed in NI-CAN Diagnostic utility, B-2

N

National Instruments CAN interfaces. *See* CAN interfaces; missing CAN interfaces.
National Instruments Web support, I-1
NC_ERR_OLD_DATA status code, 3-2
NC_ERR_OVERFLOW status code, 3-2
NC_ST_READ_AVAIL state, 3-1
NC_ST_READ_MULT state, 3-1
NC_ST_WRITE_SUCCESS state, 3-1
ncAction function, 2-9
ncConfig function, 2-9
ncCreateNotification function, 2-10
ncGetAttribute function, 3-4
ncOpenObject function, 2-9
ncRead function, 2-10
ncWaitForState function, 2-10
NI Developer Zone, I-1
NI-CAN configuration and diagnostic utilities
  failures
    Windows 2000, C-3
    Windows Me/98/95, A-3
    Windows NT, B-1
  missing CAN interface, in Windows NT, B-1
  overview, 1-14, 5-1
  starting
    LabVIEW RT, 5-3
    Windows 2000/NT, 5-2
    Windows Me/98/95, 5-1
NI-CAN Diagnostic utility, 5-1
NI-CAN error cluster (table), 2-11
NI-CAN hardware
  AT-CAN series board, 1-9
  overview, 1-9
  PCI-CAN/DS series board, 1-9
  PCMCIA-CAN series card, 1-9
  processor
    embedded processor, 1-10
  PXI-846*x* series boards, 1-9
NI-CAN object hierarchy, 1-12
  applying NI-CAN objects (figure), 1-13
  simple CAN device network application (figure), 1-12
NI-CAN software
  See also programming.
  C/C++ language interfaces, 2-2
  components
    driver and utilities, 1-14
    firmware image files, 1-15
  determining version installed
    Windows 2000, C-4
    Windows Me/98/95, A-5
    Windows NT, B-3
  error cluster (table), 2-11
  independent design, 1-11
  object hierarchy
    applying NI-CAN objects (figure), 1-13
    simple CAN device network application (figure), 1-12
  object-oriented design, 1-11
  overview, 1-11
  problem encountered
    See also troubleshooting and common questions.
    Windows 2000, C-4
    Windows Me/98/95, A-5
    Windows NT, B-3
  status code (table), 2-12
  uninstalling
    some components left installed, A-6, B-4, C-5
NI-DAQ, synchronizing with RTSI bus, 1-17
no resources assigned error, Windows NT, B-1
nondestructive bitwise arbitration, 1-2
number of devices
  high-speed CAN, ISO 11898 requirements, D-8
  low-speed CAN, ISO 11519-2 requirements, E-6

O

object hierarchy, in NI-CAN software
applying NI-CAN objects (figure), 1-13
simple CAN device network application (figure), 1-12
object-oriented design, of NI-CAN software, 1-11
objects
See also CAN Objects.
synonymous with instance, 1-11
opening objects, 2-9
operating system independence, of NI-CAN software, 1-11

P

PCI-CAN series board
and RTSI, 1-10
PCI-CAN/2 parts locator diagram (figure), D-4
power supply information for high-speed CAN ports, D-3
power source jumpers (figure), D-6
specifications, H-1
PCI-CAN/DS series board, 1-9
PCI-CAN/LS series board, 1-9
power supply information for low-speed CAN ports, E-3
PCI-CAN/LS2 series board, 1-9
parts locator diagram (figure), E-3
power source jumpers, E-5
power supply information for low-speed CAN ports, E-3
PCMCIA-CAN series card, 1-9
description of cable types, 1-10
PCMCIA-CAN cable (figure), D-2
specifications, H-2
PCMCIA-CAN/LS series card
PCMCIA-CAN/LS cable (figure), E-2
replacing termination resistors, E-12
pinout for 9-pin D-sub connector
high-speed (figure), D-1
low-speed (figure), E-1
pins used by PXI-846*x* series boards, G-3
power requirements for the high-speed CAN physical layer for bus-powered versions (table), D-6
power requirements for the low-speed CAN physical layer for bus-powered versions (table), E-5
power source jumpers (figure), D-6, E-5
power supply
bus power supply requirements
high-speed CAN, D-6
low-speed CAN, E-5
information
high-speed CAN ports, D-3
low-speed CAN ports, E-3
power requirements
high-speed CAN physical layer, bus-powered versions (table), D-6
low-speed CAN physical layer, bus-powered versions (table), E-5
power source jumpers (figure), D-6, E-5
problem solving. *See* troubleshooting and common questions.
processor
embedded processor, 1-10
programming
accessing NI-CAN software
C/C++ language interfaces, 2-2
direct entry access, 2-3
LabVIEW function library, 2-1
application examples, 4-1
CAN Network Interface Object, using with CAN Objects, 3-2
checking status of function calls
C and C++, 2-12
LabVIEW, 2-11

choosing NI-CAN Objects
CAN Network Interface Objects, 2-4
CAN Objects, 2-5
detecting state changes, 3-4
interaction of NI-CAN software with your application (figure), 1-16
LabVIEW RT, 2-1
model for NI-CAN applications
closing objects, 2-10
communicating using objects, 2-9
configuring objects, 2-9
general program steps (figure), 2-8
opening objects, 2-9
reading data, 2-10
starting communication, 2-9
waiting for available data, 2-10
queues
disabling queues, 3-2
empty queues, 3-2
full queues, 3-2
read and write queues, 3-1
state transitions, 3-1
synchronizing RTSI bus with NI-DAQ, 1-17
PXI, CompactPCI, and RTSI, G-3
PXI-8460
fault reporting capabilities, G-3
parts locator diagram (figure), E-4
port characteristics for bus-powered ports, H-2
replacing termination resistors, E-10
termination resistors
location of, (figure), E-11
preparing lead wires of, (figure), E-11
PXI-8461
fault reporting capabilities, G-3
parts locator diagram (figure), D-5
port characteristics, H-2
PXI-8462
fault reporting capabilities, G-3
PXI-846*x* series boards
and RTSI interface, 1-10, G-3
hardware overview, 1-9
pins used by, G-3

Q

questions. *See* troubleshooting and common questions.
queues
disabling queues, 3-2
empty queues, 3-2
full queues, 3-2
read and write queues, 3-1
state transitions, 3-1

R

reading data, 2-10
Real-Time System Integration (RTSI). *See* RTSI.
related documentation, *xiv*
Remote Transmit Request (RTR) field, 1-4
resistance, determining termination, E-7
resistor
termination
high-speed CAN (figure), D-8
location on PCI-CAN/LS2 board (figure), E-9
low-speed CAN (figure), E-6
preparing lead wires of replacement
PCI-CAN/LS2 (figure), E-10
PCMCIA-CAN/LS cable (figure), E-12
replacing
low-speed CAN, E-10
PCI-CAN/LS board, E-9
PCMCIA-CAN/LS cable, E-12

RTSI
AT-CAN series RTSI connector pinout, G-1
bus definition and overview, 1-16
cable, G-4
definition of, 1-10
interface description, G-1, G-3
low-speed CAN, G-2
PCI-CAN series RTSI connector pinout, G-2
pins used by PXI-846*x* series boards (table), G-3
programming, G-4
synchronization to a common trigger, 1-10
synchronizing with NI-DAQ, 1-17

S

Smart Distributed System (SDS), 1-11
SOF (Start of Frame) field, 1-3
software
overview, 1-11
specifications
AT-CAN series board, H-1
DeviceNet cable-length specifications (table), D-7
PCI-CAN series board, H-1
PCMCIA-CAN series card, H-2
standard for CAN, 1-1
Start of Frame (SOF) field, 1-3
state changes, detecting, 3-4
state transitions, queues, 3-1
status of function calls, checking
C and C++, 2-12
LabVIEW, 2-11
status code (table), 2-12
stuff error, 1-6
system integration, by National Instruments, I-1

T

technical support resources, I-1
termination resistance, determining, E-7
termination resistor
location on PCI-CAN/LS2 board (figure), E-9
placement (figure), D-8
placement for low-speed CAN (figure), E-6
preparing lead wires
PCMCIA-CAN/LS cable replacement (figure), E-12
preparing lead wires of replacement
PCI-CAN/LS (figure), E-10
replacing
PCI-CAN/LS board, E-9
PCMCIA-CAN/LS cable, E-12
termination resistors
location of, low-speed CAN (figure), E-11
preparing lead wires of, low-speed CAN (figure), E-11
troubleshooting and common questions
Windows 2000
CAN hardware problem encountered, C-3
common questions, C-4
missing CAN interface, C-2
NI-CAN Diagnostic utility failures, C-3
NI-CAN software problem encountered, C-3
problem shown in Device Manager, C-2
Windows 2000 Device Manager, C-1
Windows Me/98/95
CAN hardware problem encountered, A-4
common questions, A-5
interrupt resource conflict, A-4
memory resource conflict, A-3

missing CAN interface, A-2
NI-CAN software problem encountered, A-4
problem shown in Device Manager, A-2
Windows Me/98/95 Device Manager, A-1
Windows Me/98/95NI-CAN configuration and diagnostic utilities failures, A-3
Windows NT
CAN hardware problem encountered, B-2
common questions, B-3
interrupt resource conflict, B-2
memory resource conflict, B-2
missing CAN interface, B-2
missing CAN interface in NI-CAN configuration and diagnostic utilities, B-1
NI-CAN configuration and diagnostic utilities failures, B-1
NI-CAN software problem encountered, B-2
no resources assigned, B-1

U

uninstalling
NI-CAN software
some components left installed, A-6, B-4, C-5
using this manual set, *xiii*
utilities. *See* NI-CAN configuration and diagnostic Utilities; NI-CAN NI-CAN configuration and diagnostic utilities.

W

waiting for available data, 2-10
Web support from National Instruments, I-1
Windows 2000
troubleshooting and common questions
CAN hardware problem encountered, C-3
common questions, C-4
missing CAN interface, C-2, C-3
NI-CAN Diagnostic utility failures, C-3
problem shown in device manager, C-2
uninstalling
CAN software, C-5
Windows 2000/NT
NI-CAN driver and utilities, 1-14
starting NI-CAN configuration and diagnostic utilities, 5-1
Windows Me/98/95
NI-CAN driver and utilities, 1-14
starting NI-CAN configuration and diagnostic utilities, 5-1
troubleshooting and common questions
CAN hardware problem encountered, A-4
common questions, A-5
interrupt resource conflict, A-4
memory resource conflict, A-3
missing CAN interface, A-2, A-4
NI-CAN configuration and diagnostic utilities failures, A-3
problem shown in device manager, A-2
uninstalling
CAN software, A-6
Windows NT
troubleshooting and common questions
CAN hardware problem encountered, B-2
common questions, B-3
interrupt resource conflict, B-2
memory resource conflict, B-2
missing CAN interface, B-2

missing CAN interface in NI-CAN Configuration utility, B-1
NI-CAN configuration and diagnostic utilities failures, B-1
NI-CAN software problem encountered, B-2
no resources assigned, B-1
uninstalling
CAN software, B-4
worldwide technical support, I-2